STATE-SPACE ENCODING DRIVEN ERROR RESILIENCE IN CONTROL SYSTEMS AND CIRCUITS

A Dissertation
Presented to
The Academic Faculty

By

Suvadeep Banerjee

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering

Georgia Institute of Technology

May 2018

STATE-SPACE ENCODING DRIVEN ERROR RESILIENCE IN CONTROL SYSTEMS AND CIRCUITS

Approved by:

Dr. Abhijit Chatterjee, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Bonnie Ferri
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Arijit Raychowdhury
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Raheem A Beyah
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Jacob A. Abraham
Department of Electrical and Computer Engineering
The University of Texas at Austin

Date Approved: December 14, 2017

ACKNOWLEDGEMENTS

First and foremost, I am immensely thankful to my adviser Dr. Abhijit Chatterjee for providing me the opportunity to pursue my PhD research under his supervision. His advice, guidance and encouragement has been a constant source of motivation and inspiration throughout the course of my research at Georgia Tech. I joined Dr. Chatterjee's lab after a brief hiatus from academic quarters during which I was working in the consultancy industry. He assisted me in garnering the apposite academic knowledge for performing research by suggesting the appropriate and relevant courses required. Over the past 5 years, he has been instrumental in the development of my research acumen. He has taught me how to articulate any complex problem and solve it by separating into multiple simpler problems. I am grateful for his insight and intuition on exciting research topics, the eagerness and enthusiasm that he demonstrated while discussing crucial aspects of my research, his patience to revise and improve all my research documents and the passionate and sincere compliments he provided. Over all these years, the innumerable discussions have had a significant influence in shaping my professional and personal outlook in life. I am forever indebted to Dr. Chatterjee for this remarkable phase of my life.

I profusely thank Dr. Jacob A Abraham from UT Austin for providing valuable suggestions and insightful feedback on my research. His unparalleled knowledge on error injection experiments and the motivation he provided for incorporating the FPGA-based fault emulation models in my research shaped the central core of my thesis. I am grateful that he agreed to serve in my dissertation committee and I highly appreciate his august presence in my final defense. I sincerely thank my thesis reading committee members Dr. Bonnie Ferri and Dr. Arijit Raychowdhury for advising me on the proper structure and content of my thesis. Dr. Ferri provided pivotal suggestions about how to refine my thesis and improve my research presentations. Dr. Raychowdhury discussed critical aspects of my research for the betterment of my thesis. I also wish to acknowledge Dr. Raheem Beyah for

agreeing to serve in my dissertation committee and furnishing vital feedback on improving the security research. I thank the funding agencies of National Science Foundation (NSF) and Samsung Corporation for the sponsorship of my research during my PhD. I appreciate the initial mentorship of Dr. Hyun Choi during my first year of PhD that instilled in me the essence of research collaboration and exchange of ideas.

I sincerely thank Dr. Suriyaprakash Natarajan and Dr. Eli Chiprout for the constant guidance and support during my internships at Intel. The myriad discussions I had with them benefited me immensely in comprehending the multi-faceted issues being faced by the IC industry.

I wish to extend my sincere thanks to all my colleagues in our lab research group: Shyam Kumar Devarakond, Jayaram Natarajan, Joshua Wells, Sen-Wen Hsiao, Aritra Banerjee, Debesh Bhatta, Thomas Moon, Nicholas Tzou, Barry Muldrey, Debashis Banerjee, Xian Wang, Sabyasachi Deyati, Md Imran Momtaz, Sujay Pandey and Jun-Yang Ryan Lei. I particularly wish to thank Sabyasachi and Barry for all the productive and fruitful discussions we had over the course of my PhD research. In addition, I deeply acknowledge the assistance of Balavinayagam Samynathan in spite of his busy schedule in guiding me with the FPGA error injection experiments. I also thank Dr.Álvaro Gómez-Pau for the research collaboration during his visit to Georgia Tech and his amiable personality made it enjoyable to work with him.

I sincerely thank my friends and seniors at Georgia Tech for allowing me to withstand the tough ordeals of PhD life at USA - Arindam Khan, Abhishek Banerjee, Anshuman Goswami, Ananda Barua, Ranadip Acharya, Souryadeep Bhattacharyya, Sandeep Samal, Monodeep Kar, Sourav Dutta, Arkadeep Kumar, Swarnava Ghosh, Anish Mukherjee, Dhwanil Shukla, Samantak Gangopadhyay, Abhinav Parihar and Khondker Zakir Ahmed. Thanks are also due to the great friends I made while playing squash at Georgia Tech - Parth Agrawal, Ayush Agrawal, Rachit Agarwal, Pratik Mital, Saad bin Nasir, Samer Heiba and Vivek.

I am forever indebted to my extended family. I was lucky to have my sister Mousumi Mukherjee, my brother-in-law Santonu Mukherjee and my nephew Soham Mukherjee at Atlanta when I came to Georgia Tech in 2012. The presence of their family negated the familial hollowness that I experienced after moving to Atlanta. In addition, I am grateful to my parents-in-law Praban Palan Chatterjee and late Bratati Chatterjee. They have been instrumental in providing me encouragements and good wishes over the years. I sincerely miss the fondness my mother-in-law bestowed upon me and I know she would have been proud in sharing this accomplishment. I am also grateful to my sister-in-law Sohini Chatterjee and her sweet daughter Rai for spreading love and warmth in our lives.

Finally and most importantly, my parents Kanailal Banerjee and Ratna Banerjee endowed me with all the determination, perseverance, patience and brought me up to be a good human being. I would have never achieved my personal ambitions without their innumerable sacrifices. Their endless support and unconditional love has helped me through all the peaks and troughs of my life. It is indescribable in words how much my wife Paulami Chatterjee has done for me in my life. She had been the patient girlfriend, best friend, supportive listener and a true companion over all these years. Her calming composure and personal experiences with her own PhD had made the long-distance relationship between us possible and endurable during the course of my stay at Atlanta. Falling in love with her is the most beautiful experience of my life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

The objective of the proposed research is to develop methodologies, support algorithms and software-hardware infrastructure for detection and diagnosis of parametric failures, transient soft errors and security attacks in linear and nonlinear circuits and systems for sensing and control. This research is motivated by the proliferation of autonomous sense-and-control real-time systems, such as intelligent robots and self-driven cars, that must maintain a minimum level of performance in the presence of unavoidable electromechanical degradation of system-level components in the field as well as external security attacks. A key focus is on rapid recovery from the effects of such anomalies and impairments with minimal impact on system performance while maintaining low implementation overhead as opposed to traditional schemes for recovery that rely on duplication or triplication. Real-time detection and diagnosis techniques are investigated and rely on analysis of state-space encoding based check signatures. For on-line error detection and diagnosis in control systems, linear and nonlinear state space encodings of the system behavior are analyzed in real-time. Recovery is initiated using guided reinforcement learning algorithms that determine how best the system should be controlled in the presence of the diagnosed performance impairments. For cyberphysical systems, these state-space encodings are used to detect malicious security attacks and to diagnose the affected components swiftly. These checks are utilized for fast recovery from such attacks while avoiding catastrophic system failure. Further research in this area will pave the way for successful deployment of self-healing autonomous systems and resilient cyberphysical systems.

CHAPTER 1
INTRODUCTION

In November 1971, Intel launched the first commercial microprocessor chip, the 4004 containing 2300 tiny transistors, thus ushering the world into a technological revolution spanning decades. Since then, the expansion of computational capabilities fueled by the self-fulfilling prophecy of Gordon Moore, Intel's co-founder has enabled the integration of computing in our daily life. Moore's law predicts that processing power doubles roughly every two years as smaller transistors are packed more tightly onto silicon wafers, boosting performance and reducing costs. The attempt to satisfy this rule has achieved exponential progress in computing performance that is orders of magnitude superior than the 1971 processor. This type of meteoric advancement in technology is hitherto unobserved in any domain of scientific development. A modern Intel Coffee Lake processor contains more than 2 billion transistors and delivers 500,000 times as much computational capability as a 1971 processor. The abundance of computing power can be perceived from the fact that a modern smartphone is more capable than a room-sized supercomputer in the 1980s. The overwhelming march of computing progress has dramatically altered every phase of human life with promises of sweeping improvement in the future days.

After almost five decades of relentless pursuit of Moore's law, imminent signs of slowdown of the exponential performance achievement are being observed [1–6] as shown in Figure 1.1 while posing a question over the future of computing progress. Until the 2000s, geometric scaling of transistor feature sizes were sufficient to maintain the doubling of transistor counts in a chip, thus reaping the obtained performance benefits. After the 2000s, continuing further geometric scaling posed numerous problems and the semiconductor industry kept devising sophisticated technical measures to keep pace with the predictions of Moore's law [7] - at 90 nm, strained silicon was introduced; at 45 nm, new materials

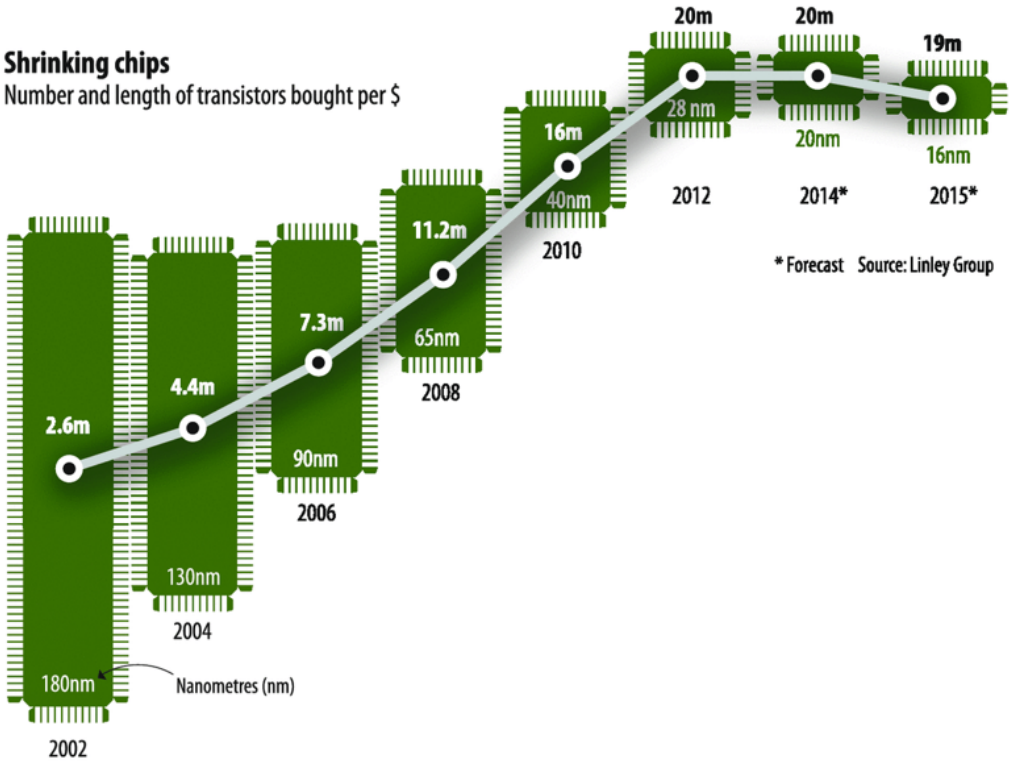


Figure 1.1: The slowing down of Moore's law is evident from the decrease in the number of transistors bought per $, thus indicating rising costs

(high-κ dielectric) to increase the capacitance of each transistor layered on the silicon were introduced; at 22 nm, trigate transistors [8] maintained the scaling. At present, three major problems hinder the progress of feature scaling along Moore's law. Firstly, the photo-lithography process used to transfer the chip patterns to the silicon wafer needs significant technological advancement for enabling further feature scaling. Presently, light sources with 193nm wavelength are used to generate feature sizes of 14 nm, adding extra complexity and cost to the manufacturing process with each generation of ICs. Though extreme UV [9] with 13nm wavelength is capable of solving this problem, production-ready EUV technology has proven difficult to engineer. Secondly the extent of feature size scaling possible is unclear; at 2 nm transistors will be a few atoms wide and device physicists think it's unlikely that they will operate reliably at such a small scale. Thirdly, even if the modernization of photo-lithography process is adopted along with resolving the question of scaling, the

specter of power usage and energy dissipation looms large. As higher number of transistors are tightly packed into an IC, dissipation of the generated heat becomes increasingly harder creating power densities as high as values found in a rocket nozzle. A compounding of all these factors has stopped the clock speeds of modern processors from increasing any further and thwarted the benefits that workloads receive from ever-increasing core frequency. Though researchers are exploring several alternatives to silicon based MOSFETs such as carbon nanotubes [10–15], graphene [16–20], spin-based devices [21–25], none has been declared a prominent solution. This has allowed the future of computing to be defined by improvements in two different areas, beyond raw hardware performance.

1. Artificial Intelligence: Artificial intelligence (AI) is defined as a framework where machines learn and make decisions in an intelligent, human-like way. The teaching of computers to do complex tasks is enabled through machine learning that is hinged on the basic idea that software can simulate the activity in layers of neurons in the neocortex, the hub of complex decision-making in the human brain. Such an algorithm learns to recognize patterns in digital representations of sounds, images, and other data. One branch of AI has shown considerable promise in solving highly complex tasks - 'Deep Learning'. Deep learning is a class of machine learning algorithms that use a cascade of multiple layers of nonlinear processing units for feature extraction and transformation of input data. One fascinating success of deep learning was observed in early 2016 when AlphaGo, a Google's deep-learning based computer algorithm crushed the current world champion, Lee Sedol in a game of Go [26–28]. This phenomenal achievement can be put to context while discussing the complexity of the above feat. In a game of Go, there are more possible board positions than there are particles in the observable universe. As a result, an AI-enabled Go-playing system cannot simply rely on computational brute force, provided by Moore's law, to prevail. The success of 'deep learning' technology demonstrates that huge performance gains can be achieved through new algorithms. Slowing progress in hardware

innovations will provide stronger incentives to develop better software as outlined by the recent IEEE CS 2022 report [29] from the IEEE Computer Society. At the foundation of the report is the understanding that by 2022, intelligence will become seamless and ubiquitous in all forms of technology. Machine intelligence has already started transforming every sphere from communications and computing to medicine, manufacturing, and transportation and will increasingly integrate with our daily lives.

2. Computing Architectures: The second area of improvement lies in novel computing architectures that can create specialized chips optimized for specific jobs. As computation becomes increasingly data centric and the scalability limits in terms of performance and power are being reached, alternative computing paradigms different from the conventional von Neumann architecture are being sought. Now, chips are being designed specifically for cloud computing, neural-network processing, computer vision and other tasks. Novel architecture paradigms are being explored for achieving the performance boost obtained from Moore's law scaling for so long. Three different types of architectures have showed early promise:

   i) *In-memory Computing* - Scientists have recently explored the concept of 'in-memory computing' [30] that is expected to yield 200x improvements in computer speed and energy efficiency. Unlike the conventional von Neumann architecture, this concept uses phase-change memory (PCM) for both storing and processing data at the same physical location. Such co-existence of computation and storage at the nanometer scale removes the bottleneck of constant data communication between memory and the computing unit, thus paving the way for building ultra-dense, low-power and massively-parallel computing systems.

   ii) *Quantum Computing* - Unlike binary digital electronic computers where data is encoded into binary states, quantum computing aims to exploit quantum-mechanical phenomena such as superposition and entanglement to form quantum bits that can be

superpositions of individual states. Such quantum effects enable the quantum bits or "qubits" to occupy multiple states simultaneously, thus promising to tackle problems that are impossible to solve by brute-force computations. The research community [31] is heavily investing in exploring the physical implementation and numerical capabilities of such quantum computers.

iii) *Cognitive Computing* - Cognitive computing [32] attempts to build computer chips that use a network of "neurosynaptic cores" to manage information in a way that resembles the functioning of neurons in a brain. Instead of using the sequential programming based von Neumann architecture, these cognitive computers stores and processes information in a distributed, parallel way, like the neurons and synapses in a brain. Researchers are attempting to create computers that have about half the complexity of human brain while consuming minimal power, in the orders of kilowatts.

With significant research efforts attempting to drive the growth of computing along different avenues, the semiconductor industry is currently predicting that artificial intelligence and machine learning enabled applications will pervade all domains of human life. This dissertation develops algorithms and methodologies for enabling the vision of such a future as described next in the motivation.

1.1 Motivation

The motivations of this research stem from the efforts to tackle system malfunctions in autonomous and cyber-physical systems due to errors induced by component failures or malicious security attacks:

1.1.1 Autonomous Systems

The successful deployment and assimilation of the intelligent autonomous systems in human society depends on the trustworthiness of these systems. Unless these centralized and

distributed autonomous systems [33–41] are *dependable* (*reliable, fault-tolerant, available, maintainable, safe* and *secure*), such systems may never see fruition in the commercial arena or be useful for critical operations. This is because many such emerging applications can have life-threatening consequences if they malfunction in the field. Self-driven cars are great examples of this paradigm. While proponents of such systems argue that automation will make self-driven cars safer than human-driven vehicles, they ignore the fact that the controlling hardware and software itself may have bugs, that sensors and actuators can degrade over time and that control principles will need to adapt in real-time to account for electrical and mechanical wear-and-tear. For reliable and dependable vehicle operation, transformational cross-layer (sensor, actuator and control) design methods need to be developed that deliver ultra-high levels of dependability (compared to the safety/dependability of aircraft and space travel), on a per-vehicle basis to thousands of autonomous vehicles for the technology to ever become commercially successful on a massive scale. A single vehicle malfunction resulting in injury or loss of life can spell doom for the commercialization of such technology in the future. In this context, the control operations of modern autonomous systems are evolving towards electronics as opposed to mechanical/hydraulic means of control as evident from the proliferation of "steer-by-wire" and "brake-by-wire" [42–47] control systems. While this improves the reliability of the system due to use of fewer mechanical components, the overall dependability of the system is now contingent on error-free operation of the control electronics, sensors and actuators. Detection and mitigation of such control malfunction in real-time forms the core objective of this research.

1.1.2 Cyber-physical systems

Modern cyber-physical systems (CPSs) are complex combinations of physical processes, computational resources and communication capabilities. These cyber-physical systems are permeating every sphere of modern life in various domains such as energy production, health care and telecommunications. Examples of cyber-physical systems include

smart grid, chemical processes and medical monitoring. Critical infrastructure such as water treatment plants, power generation systems and transportation networks are integrating cyber-technologies with physical processes to improve operational efficiency, resource scalability and system autonomy. The geographical separation between sensing and actuation in these systems enforces the use of network communication for effective control. At the same time, the increased adoption of cyber-capabilities introduces vulnerabilities that undermine the reliable operation of these systems. Real-world cyber-security incidents such as the Queensland sewage treatment plant attack in 2000 [48, 49], US petrochemical process attack in 2003 [50], Daimler-Chrysler auto manufacturing plant attack in 2005 [51], Baku oil pipeline attack in 2008 [52], California pipeline attack in 2009 [53], Stuxnet attack in Iranian uranium-enrichment plant in 2010 [54–56], German steel plant attack in 2014 [57] and Ukrainian power grid attack in 2015 [58] demonstrate that these cyber-physical systems are prone to failures due to attacks on the physical infrastructure and communication management. These cyber-physical systems need to be dependable (*reliable, fault-tolerant, available, maintainable and secure*) for their ubiquitous deployment across all domains of life. Hence, it is imperative to develop robust and effective real-time attack monitoring and threat mitigation mechanisms.

The primary motivations of this dissertation are:

- To develop fundamental paradigms for ensuring the safety and dependability of autonomous systems of the future that use linear and nonlinear control systems for real-time operation (e.g. autonomous vehicles). Such paradigms will detect, diagnose and correct for the effects of transient errors or parametric deviations with low latency while generating prognostics for off-line repair

- To develop fundamental security paradigms that can detect, diagnose and mitigate the effects of external attacks on the operations of cyberphysical systems, while achieving very low attack detection latency and extremely high coverage of security attack

mechanisms. It is assumed that attacks can be triggered through wireless interfaces with the cloud or by direct wireless attack on the data transmissions

1.2 Prior Work

Dealing with failures has been a major concern even with the earliest electronic systems [59, 60]. The use of checksums for failure tolerant matrix arithmetic and signal processing algorithms was first investigated by Hua, Jou and Abraham in [61, 62] and led to the field of *algorithm-based fault tolerance* (ABFT) in the context of complex signal processing algorithms. The underlying research focused on exact detection and correction of errors in classes of widely used signal processing algorithms such as for matrix arithmetic and the Fast Fourier Transform, among others. In [63], a methodology for soft error detection and correction called *algorithmic noise tolerance* (ANT) was developed and further explored in [64–66]. Here, the focus was on using reduced-precision duplication codes to detect and correct errors with the understanding that not all errors need to be detected and corrected precisely and that "perfect" error compensation is not always required for various classes of signal processing algorithms such as for video compression. A class of codes called "real-number checksums" for matrix-vector computations was developed by Nair and Abraham in [67]. This class of codes was used by Chatterjee and d'Abreau in [68] to perform error detection and correction in linear digital state variable systems. While error detection could be performed with low overhead in [68], error correction incurred large overheads both in area and time using the proposed algorithms. To resolve this, probabilistic and guided error compensation schemes were proposed in [69–72]. In [73], continuous checksums were used for the first time to detect and correct faults in linear analog state variable systems. The state matrix was encoded using checksums and by tracking the system state variables, any change in the analog transfer function of the system could be detected in real-time. The continuous checking methodology was extended to use state estimation techniques in [74]. However, prior work primarily focused on concurrent error detection in analog and digital

state variable systems and does not apply directly to control systems, as the latter consist of *plant* and *controller* equations that must be combined together in a meaningful way to allow detection of perturbations in the plant as well as its controller.

The problem of ensuring correct system operation under malfunctions and external disturbances is typically tackled by the control community through robust control and fault-tolerant control. Negative feedback control is inherently robust enough to suppress small faults in the actuators or plant. However, even small faults or disturbances in the sensors of a feedback control system lead to potentially disastrous consequences due to immediate deviation of the response trajectory from nominal behavior. Robust control methods address this by explicitly designing mechanisms to deal with control uncertainty within specified bounds. However, Wang and Zhang demonstrated the fundamental shortcomings of robust control methods such as H_∞ loop shaping [75–78] and Lyapunov-based controllers [79] in [80]. These methods are capable of handling limited parameter uncertainties and suitable for slow-varying systems such as chemical processes. Robust controller design is difficult for future complex cyber-physical systems operating in critical environments (such as self-driven cars) where wide range of anomalies and system degradations are possible. Fault tolerant control aims to avoid these system failures by implementing sophisticated methodologies that can broadly be classified into 4 main categories:

1) Analytical Redundancy - This methodology implements a process model in software [81] for constant comparison with the actual system data. The duplicated process model is provided with same input signals as the original system at each time instant. The error signal is formed from the comparison between process model output and actual system output. In the presence of errors, this error signal is non-zero indicating faults in the system. Though this approach has high fault coverage, it involves analytical duplication of the process model that becomes infeasible for highly complex systems due to two main reasons - i) accurate process model over the entire feasible state space may not be available for future cyber-physical systems and ii) duplication of system resources is extremely expensive for

large applications.

2) Limit Checking - Signal based fault detection methods [82] employ limit value checking and plausibility ranges on measured signals for detection of errors and disturbances. Pre-computed thresholds are determined for each signal based on nominal fault-free simulations. In the presence of errors, detection is accomplished if the measured signals cross the established bounds. Though this technique is easy to implement and has the least overhead, the failure coverage is low for non-catastrophic causes such as component degradation or transient errors that are essential for continuous prognosis in cyber-physical systems.

3) Residual Generators - Residual generators are popular choices for fault detection in the control community and these can be broadly classified into two main categories - i) observer based approaches [83, 84] and parity-space-like approaches[85, 86]. In this scheme, a specially designed residual filter is employed that generates a failure map from the system measurements. For each fault, the failure map creates unique signatures that are zero in the absence of any errors. This methodology has been proved to have high fault coverage and diagnostic capabilities. However, the primary drawback of such a scheme is the assumption of prior knowledge of failure modes that are utilized in the design of the residual generator. Increase in the cardinality of feasible fault set results in such high computational overhead that this approach is impractical for online implementation on embedded processors. Furthermore, previous research has mostly focused on the application of residual generators to linear systems [87, 88], weakly nonlinear systems and strongly nonlinear systems operating around an equilibrium point making these techniques unscalable for complex systems operating with a high degree of nonlinearity.

4) PCA-based Detection - Principal component analysis (PCA) has previously been used for fault detection in complex processes [89]. Under this scheme, the eigenstructure of the covariance matrix of collected data is computed under fault-free operating conditions. This allows the characterization of high-dimensional process data in a lower dimensional

representation space, thus revealing linear relations among system states. The PCA model projects the measurements into a residual space orthogonal to the model subspace, thus enabling detection of parametric deviations. While PCA-based methodologies have been effective for fault detection, it has been pointed out [90] that the underlying models suffer from outlier sensitivity and the fault coverage varies widely due to model order truncation.

This dissertation addresses the insufficiency of the above methodologies for error detection and correction in linear and nonlinear systems operating under arbitrary failure mechanisms.

1.3 Contributions of Dissertation

The key contributions of this dissertation are summarized below:

1. State-space encoding based checking methodologies are developed for error detection in a wide class of linear and nonlinear control systems [91]. Typically, a nonlinear system switches across different operating points around which the system behavior is weakly nonlinear [92]. Prior methods have focused on system operation around fixed equilibrium points where the nonlinear behavior is limited and linearization techniques can be applied. The proposed research in this dissertation makes no such assumption and is applicable over a broad class of systems across all operating modes - i) linear, ii) weakly nonlinear and iii) strongly nonlinear.

2. This research proposes a hybrid architecture where actor-critic reinforcement learning control is used as an augmentation to the classical control algorithms to enable real-time adaptability with low latency. The state-space encoding methodologies are exploited for efficient and rapid self-learning of the optimal control laws in presence of system malfunctions (sensors, actuators and electro-mechanical component performance degradation) to restore system level performance as best as possible.

3. The diagnostic information extracted from the state encoding based error signal is

utilized to significantly accelerate the self-learning process of the actor-critic reinforcement control through bootstrapping the learning process.

4. The proposed methodology is capable of detecting and correcting errors created due to arbitrary failure mechanisms. Previous techniques assume pre-determined fault models in the design of the detection algorithms. The proposed approach is not predicated on any such assumption and can be applied to a nonlinear system amenable to arbitrary failure mechanisms.

5. This research presents a cross-layer fault model for the first time in which environmental anomalies, sensor/actuator malfunctions and soft errors in control program execution are studied. The developed methodologies have high coverage in detection of failures across different layers of abstraction.

6. Demonstration of the error detection and real-time control law adaptation capabilities are shown on an actual self-balancing robot.

7. These state encoding based checking schemes are modified in a manner such that malicious and engineered security attacks on distributed cyber-physical systems can be identified. Under reasonable assumptions of resources and knowledge of an attacker, it is shown that these methodologies are resilient to adversarial counter-attacks and cannot be cracked easily. The distributed checking methodology is also exploited to enable fast diagnosis of attacked sensors and actuators so that recovery from these attacks can be achieved with low latency. Simulation results on a set of IEEE benchmark power systems as well as very large scale power grids are demonstrated to emphasize the benefits of this research.

The research contributions of this dissertation are summarized in Figure 1.2.

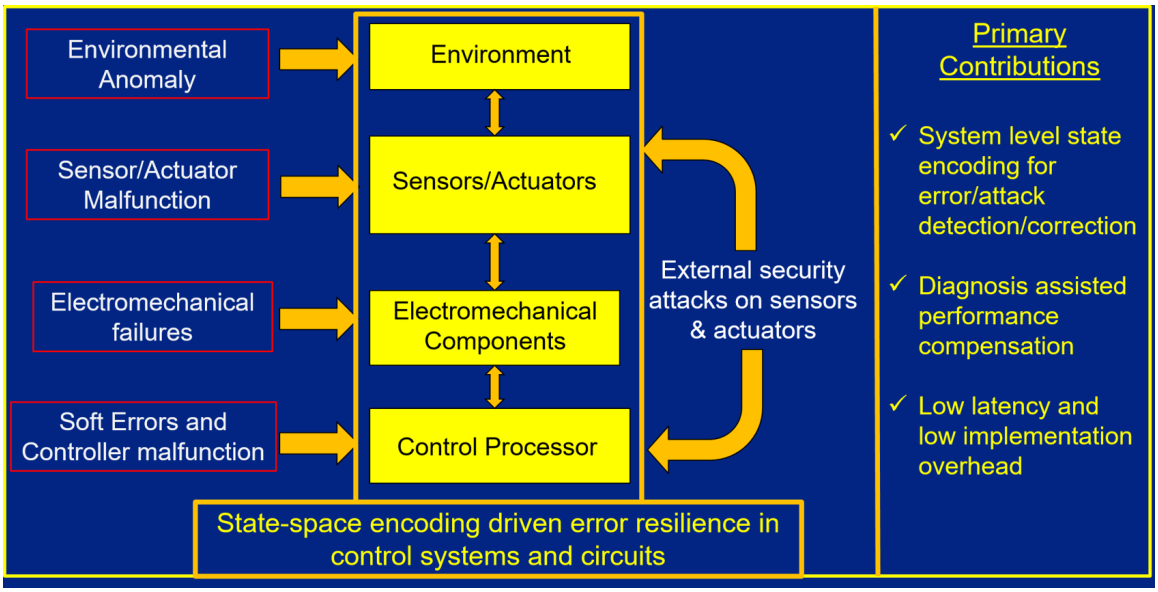


Figure 1.2: Research contributions of dissertation

1.4 Dissertation Overview

The primary objective of this dissertation is to develop methodologies and algorithms for real-time detection and correction of parametric failures, transient soft errors and security attacks in linear and nonlinear systems.

Chapter 2 introduces the foundational theory of system level state-space encoding for detection of parametric failures and transient errors for linear control systems. It is also demonstrated how such a linear encoding can be used for error correction with low latency. Simulation results on a linear control system are presented to illustrate the detection and correction capabilities.

Chapter 3 extends the encoding scheme for error detection and correction in circuits with state-space representations. Real-time noise cancellation is demonstrated on linear analog filters with two different learning-assisted adaptive schemes. Hardware experiments on a prototype circuit are presented to show the real-time noise cancellation capabilities. The encoding scheme is slightly modified for error detection in nonlinear filters and results for a nonlinear Volterra filter are discussed.

Chapter 4 proposes a real-time error detection methodology using machine learning assisted state-space encodings for nonlinear control systems operating under arbitrary failure mechanisms. For the first time, a cross-layer fault model in nonlinear control systems is assumed where failures and anomalies are assumed to occur in any section of a real system implementation. Fault injections on an FPGA and failure detection on a hardware platform along with simulation results from microarchitectural experiments comprehensively establish the benefits of the proposed detection methodology.

Chapter 5 proposes ALERA, an abbreviation for Accelerated Learning Enabled Reinforcement Architecture for real-time adaptability in nonlinear control systems. State-space encodings are utilized to accelerate the real-time learning process for control adaptation in presence of errors. Simulation results and hardware demonstration illustrate the promising potential and viability of the proposed scheme.

Chapter 6 presents a unified framework for detection, diagnosis and correction of security attacks in cyber-physical systems (CPS). Electric power grids are studied as examples of CPS along with sophisticated attack strategies. State-space encodings are redefined in a hierarchical algorithm for attack diagnosability in distributed system. The resilience of the proposed scheme is studied for adversarial counter-attack possibilities. Extensive simulation data are presented on IEEE benchmark test cases along with very large power grids to show the benefits compared to state-of-the-art methods.

Finally, chapter 7 presents the conclusions of this dissertation and recommends avenues for future research.

CHAPTER 2

CHECKSUMS IN LINEAR CONTROL SYSTEMS

This chapter introduces the theory of checksum encoding for detection and correction of errors in linear control systems. Simulation results on a linear servo motor are presented to illustrate the concepts.

2.1 Theory: System level checksum encoding

2.1.1 Error Detection

A generalized theory is developed for applying analog checksum codes [73] to real-time control system checking. Consider the state transition and control laws for a linear control system as shown in Figure 2.1. All the signals of Figure 2.1 represent vectors of elements and the circles represent adders/subtractors. For linear time-invariant systems, the matrices A, B, C, D, etc. are constant and time-invariant. The continuous time state differential equation of the plant is $\dot{x}(t) = Ax(t) + Bu(t) + v_p(t)$, where $x(t)$ is the state of the plant and $u(t)$ is the input variable. The observed variable $y(t)$ is given by $y(t) = Cx(t) + v_m(t)$. The variables $v_p(t)$ and $v_m(t)$ represent noises (perturbations) in the state and measurement vectors, respectively. The controlled variable is $z(t) = Dx(t)$ and the reference variable $r(t)$ is a stochastic process of the same dimension as the controlled variable $z(t)$. Consider the case where the closed loop controller is a linear differential system with the reference variable $r(t)$ and the observed variable $y(t)$ as inputs and the plant input $u(t)$ as output. The state differential equation of the closed loop controller is of the form $\dot{q}(t) = Lq(t) + K_r r(t) - K_f y(t)$, while the output equation of the controller is of the form $u(t) = Fq(t) + H_r r(t) - H_f y(t)$. The index r refers to the reference variable and the index f to feedback.

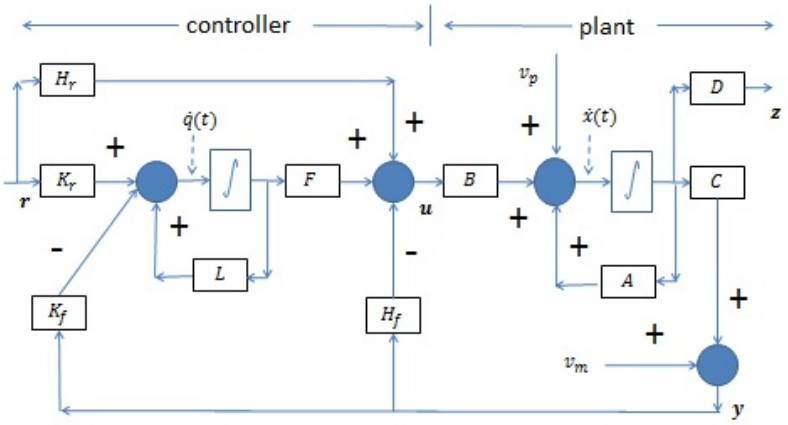


Figure 2.1: Closed-loop linear control system

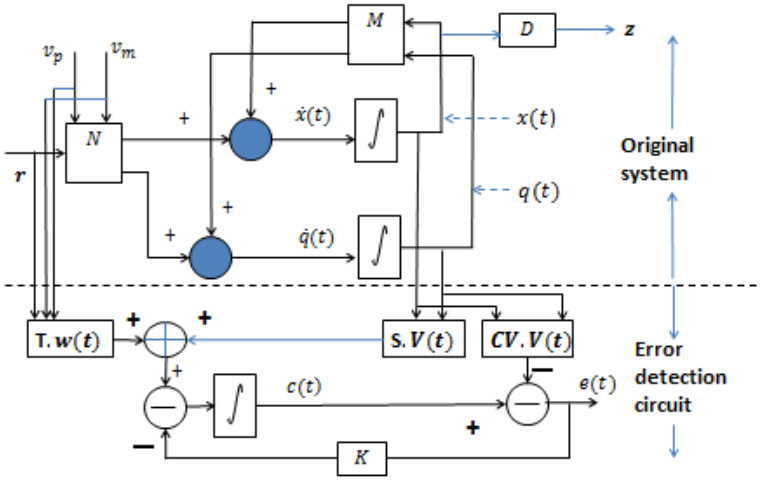


Figure 2.2: Complete System with Error Checking Block

Consider the vector $V(t) = \begin{bmatrix} x(t) \\ q(t) \end{bmatrix}$ for a *linear time-invariant* system. The differential equations for the agent and its controller are combined as follows:

$$\dot{V}(t) = MV(t) + Nw(t) \tag{2.1}$$

where $w(t) = \begin{bmatrix} r(t) \\ v_p(t) \\ v_m(t) \end{bmatrix}$ and M, N represent linear transformations of $V(t)$ and the exter-

nal input parameters $w(t)$, respectively. Let the dimensions of $V(t)$ be $n \times 1$ and let the dimensions of $w(t)$ be $m \times 1$. Then M is an $n \times n$ matrix and N is an $n \times m$ matrix of real numbers. Using some mathematical manipulation, the output equation $z = Dx(t)$ can be subsumed into the differential equation described above and errors in the same, without loss of generality, can be covered by the discussion below.

The equation $\dot{V}(t) = MV(t) + Nw(t)$ represents a linear state variable system, and real number codes [67] can be used to encode the state vector $V(t)$, using one or more check variables. These check variables can be used for error detection and correction in the operation and control of the plant [68, 69]. Each row i of the M and N matrices is scaled using a real value of magnitude α_i. Let the coding vector be the vector containing the relevant weighting factors for the ith rows of M and N, i.e $\alpha = \begin{bmatrix} \alpha_1 & \alpha_2 & \dots & \alpha_n \end{bmatrix}$. Then S is defined as $S = \alpha \times M$. Similarly T is defined as $T = \alpha \times N$. Integrating $\dot{V}(t) = MV(t) + Nw(t)$ with respect to time on both the left and right hand sides, yields $V(t) = M \int_0^t V(t) + N \int_0^t w(t)$. A check variable $c(t)$, corresponding to the coding vector α is computed as $c(t) = S \int_0^t V(t) + T \int_0^t w(t)$. In matrix form, the plant, controller and checking equations can be written jointly as:

$$\begin{bmatrix} V(t) \\ c(t) \end{bmatrix} = \begin{bmatrix} M \\ S \end{bmatrix} \int_0^t V(t) + \begin{bmatrix} N \\ T \end{bmatrix} \int_0^t w(t) \tag{2.2}$$

From this representation it is easily shown that if there is no error in the system, then $c(t) = \alpha.V(t)$. This leads us to the definition of error signal as

$$e(t) = c(t) - \alpha.V(t) \tag{2.3}$$

If a method is implemented to compute $c(t) - \alpha.V(t)$, then the output of this block $e(t)$, called the error checking block, is always zero under fault-free conditions. The implemen-

tation of the complete system with the error checking block is shown in Figure 2.2. Note that $e(t)$ is zero in the absence of any error and is fed back to the integrator with a gain of K for stability of the integrator in the error detection flow graph. In practice, a small value of K is selected so that the frequency response of the error signal $e(t)$ is not affected significantly by the feedback path.

A key feature of the error checking scheme of Figure 2.2 is that its overhead is $\frac{1}{n}$ since only one additional state is computed to check the behavior of n states. This overhead decreases as a percentage of the overall implementation cost as the complexity of the plant (agent) increases. Any malfunction in the state equations of the *agent* as well as in the state equations of the *controller* or the *checking subsystem* of Figure 2.2 is detected in real-time using continuous-time sensing. This includes malfunctions in actuators, sensors and measurement instrumentation. While the system of Figure 2.2 shows a single checksum-based checker, it is possible to determine whether the error is in the agent/controller or the checking subsystem using *two* checksum-based checkers (corresponding to a distance-3 checksum code) [73].

2.1.2 Error Compensation

It has been shown for analog circuits in [73] that if the error signal $e(t)$ is fed back with high enough gain into the erroneous state of the feedback system, the errors can be exactly compensated. To demonstrate that, for the sake of simplicity, let us assume that there is only 1 input to the system, i.e. $w(s)$ is a scalar quantity and $v_m(s)$ and $v_p(s)$ are ignored for this analysis. Hence, N is an $n \times 1$ vector. The Laplace variable of s is dropped for the purpose of legibility. Let's assume that due to parametric errors in the system, state V_i of the system is perturbed. To reflect the state perturbation, the corresponding entries of the matrices $\boldsymbol{M}$ and $\boldsymbol{N}$ are modified as

$$M_{ij} \rightarrow M_{ij} + \theta_{ij} \quad \forall 1 \leq j \leq n$$

$$N_i \rightarrow N_i + \gamma_i$$

Thus the new system matrix $\boldsymbol{M'}$ and the input matrix $\boldsymbol{M'}$ can be written as

$$\boldsymbol{M'} = \boldsymbol{M} + \boldsymbol{\theta}$$

$$\boldsymbol{N'} = \boldsymbol{N} + \boldsymbol{\gamma}$$

where the perturbation matrices are given as

$$\boldsymbol{\theta} = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ \theta_{i1} & \theta_{i2} & \cdots & \theta_{in} \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} \qquad \boldsymbol{\gamma} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ \gamma_i \\ \vdots \\ 0 \end{bmatrix}$$

Rewriting equation (2.3) as

$$\begin{aligned} e &= c - \boldsymbol{\alpha V} \\ &= c - \boldsymbol{\alpha}\left[\boldsymbol{M'}\frac{\boldsymbol{V}}{s} + \boldsymbol{N'}\frac{w}{s}\right] \\ &= c - \boldsymbol{\alpha}\left[\left(\boldsymbol{M} + \boldsymbol{\theta}\right)\frac{\boldsymbol{V}}{s} + \left(\boldsymbol{N} + \boldsymbol{\gamma}\right)\frac{w}{s}\right] \\ &= c - \boldsymbol{\alpha}\left[\left(\boldsymbol{M}\frac{\boldsymbol{V}}{s} + \boldsymbol{N}\frac{w}{s}\right) + \left(\boldsymbol{\theta}\frac{\boldsymbol{V}}{s} + \boldsymbol{\gamma}\frac{w}{s}\right)\right] \end{aligned}$$

Now by construction of the checking circuit, the term $c - \boldsymbol{\alpha}\left(\boldsymbol{M}\frac{\boldsymbol{V}}{s} + \boldsymbol{N}\frac{w}{s}\right)$ is equal to

zero. Thus the error signal shows a value of

$$e = -\boldsymbol{\alpha}\left(\boldsymbol{\theta}\frac{\boldsymbol{V}}{s} + \boldsymbol{\gamma}\frac{w}{s}\right) \tag{2.4}$$

Next it will be proved that if this error signal (output of checking circuit) is fed back to the state V_i of the system with high enough gain, the state will be properly compensated.

From equation (2.1), it is clear that the error in state V_i is given as

$$\Delta V_i = \sum_{j=1}^{n} \theta_{ij}\frac{V_j}{s} + \gamma_i\frac{w}{s} \tag{2.5}$$

Thus an accurate compensation scheme will exactly compensate for this error of the state V_i by providing $-\Delta V_i$ as feedback into state V_i. Let's assume that the error signal e is fed back to state V_i with a gain of ρ. Under this situation, the error signal will be given as, from equation (2.3),

$$\begin{aligned}
e &= c - \boldsymbol{\alpha}\boldsymbol{V} \\
&= c - \sum_{j=1, j\neq i}^{n} \alpha_j V_j - \alpha_i(V_i + \rho e) \\
&= c - \sum_{j=1, j\neq i}^{n} \alpha_j V_j - \alpha_i V_i - \alpha_i \rho e \\
e(1 + \alpha_i\rho) &= c - \sum_{j=1, j\neq i}^{n} \alpha_j V_j \\
&\quad - \alpha_i \left[\sum_{k=1}^{n} \left(M_{ik} + \theta_{ik}\right)\frac{V_k}{s} + \left(N_i + \gamma_i\right)\frac{w}{s}\right]
\end{aligned}$$

where M_{ik} and N_i are the elements of the matrices $\boldsymbol{M}$ and $\boldsymbol{N}$.

Rearranging,

$$e(1+\alpha_i\rho) = c - \sum_{j=1,j\neq i}^{n} \alpha_j V_j - \alpha_i \left[\sum_{k=1}^{n} \left(M_{ik}\frac{V_k}{s} \right) + N_i\frac{w}{s} + \sum_{k=1}^{n} \left(\theta_{ik}\frac{V_k}{s} \right) + \gamma_i\frac{w}{s} \right]$$

Now, from equation (2.1), it is known that the term $\sum_{k=1}^{n} \left(M_{ik}\frac{V_k}{s} \right) + N_i\frac{w}{s}$ is equal to the actual correct value of the state V_i, which is denoted by $\hat{V}_i$. Thus,

$$e(1+\alpha_i\rho) = c - \sum_{j=1,j\neq i}^{n} \alpha_j V_j - \alpha_i\hat{V}_i - \alpha_i \left[\sum_{k=1}^{n} \left(\theta_{ik}\frac{V_k}{s} \right) + \gamma_i\frac{w}{s} \right]$$

Again, it has been mentioned earlier that the term $c - \sum_{j=1,j\neq i}^{n} \alpha_j V_j - \alpha_i\hat{V}_i$ is equal to zero. So the expression can be simplified as

$$e(1+\alpha_i\rho) = -\alpha_i \left[\sum_{k=1}^{n} \left(\theta_{ik}\frac{V_k}{s} \right) + \gamma_i\frac{w}{s} \right] \tag{2.6}$$

Replacing the term in brackets as ΔV_i from equation (2.5) and taking $\lim_{\rho\to\infty} e(1+\alpha_i\rho)$,

$$\alpha_i\rho e = -\alpha_i\Delta V_i$$

$$\text{or,} \quad \rho e = -\Delta V_i$$

Thus it can be seen that the compensation signal ρe fed back into the state V_i is exactly equal and opposite to the error in the state V_i and is thus exactly compensated for.

2.1.3 State Detection

It has been shown that if the compensation signal is fed back to the corrupted state, the effects of the fault in the state are properly compensated. However, it is necessary to determine the corrupted state in order for failure compensation to work correctly. If the error signal is fed back to the wrong (fault-free) state, additional errors are introduced into an otherwise correct state. It can be easily shown from (2.6) that if the error signal is fed back to state V_j with the original error in state V_i, the error signal e_{ij} can be derived as

$$e_{ij} = -\frac{\alpha_i}{1 + \rho\alpha_j}\Delta V_i \tag{2.7}$$

So, it is clear that $\lim_{\rho\to\infty} e_{ij} = 0$ and thus irrespective of the state in which the error is fed back, the error line always settles to zero due to the way the error line is constructed. Thus, a second error line is needed for detecting whether the compensation is performed correctly or not. Let us assume the existence of a second error line $\varepsilon = d - \boldsymbol{\beta V}$ where the check variable d is similarly constructed as c with a different set of coding vector $\boldsymbol{\beta}$, yielding $d = 0$ under non-faulty conditions. Let us assume that ε_{ij} is the second error line value when the first error line e_{ij} is fed back to state V_j with error in state V_i. So, if error is fed back correctly to V_i, i.e. $i = j$, then

$$\begin{aligned}
\varepsilon_{ii} &= d - \sum_{j=1, j\neq i}^{n} \beta_j V_j - \beta_i(V_i + \Delta V_i + \rho e_{ii}) \\
&= d - \sum_{j=1}^{n} \beta_j V_j - \beta_i(\Delta V_i + \rho e_{ii}) \\
&= -\beta_i(\Delta V_i + \rho e_{ii})
\end{aligned}$$

Replacing the value of e_{ii} from (2.18), we get

$$\begin{aligned}\varepsilon_{ii} &= -\beta_i \Delta V_i \left(1 - \frac{\rho\alpha_i}{1+\rho\alpha_i}\right) \\ &= -\frac{\beta_i \Delta V_i}{1+\rho\alpha_i}\end{aligned}$$

Thus, for high enough gain $\lim_{\rho\to\infty} \varepsilon_{ii} = 0$, and thus the second error line goes to zero for correct compensation. However, it will be shown that for incorrect compensation to any other state V_j, though the first error line e_{ij} still goes to zero, the second error line ε_{ij} shows a non-zero value. We have,

$$\begin{aligned}\varepsilon_{ij} &= d - \sum_{k=1,k\neq i,j}^{n} \beta_k V_k - \beta_i(V_i + \Delta V_i) - \beta_j(V_j + \rho e_{ij}) \\ &= d - \sum_{k=1}^{n} \beta_k V_k - \beta_i \Delta V_i - \beta_j \rho e_{ij} \\ &= -\beta_i \Delta V_i + \frac{\beta_j \rho \alpha_i}{1+\rho\alpha_j} \Delta V_i \\ &= \frac{\rho(\beta_j\alpha_i - \beta_i\alpha_j) - \beta_i}{1+\rho\alpha_j} \Delta V_i\end{aligned}$$

Thus, it is seen that the limiting value is given as

$$\lim_{\rho\to\infty} \varepsilon_{ij} = \frac{\beta_j\alpha_i - \beta_i\alpha_j}{\alpha_j} \Delta V_i \tag{2.8}$$

For incorrect compensation, the second error line ε_{ij} shows a non-zero value and by observing ε_{ij}, the correct state to compensate is determined. Ensuring that the expression in (2.19) doesn't equate to zero requires that

$$\begin{bmatrix}\alpha_i \\ \alpha_j\end{bmatrix} \neq k \begin{bmatrix}\beta_i \\ \beta_j\end{bmatrix} \qquad \forall k \in \mathbb{R}, \quad i,j \in [1,n] \quad i \neq j \tag{2.9}$$

Once a coding vector $\boldsymbol{\alpha}$ is selected, there are infinite choices of $\boldsymbol{\beta}$ satisfying (2.9). In the next chapter, we explore two different schemes for diagnosis of corrupt states using a real-time hardware-directed "learning" procedure.

2.2 Application: Position control by a servo motor

The proposed checking algorithm is demonstrated on a feedback control system that positions an inertial load using a servo motor. Starting from rest, the control problem is to rotate the inertial load by an angle θ with maximum accuracy, in the least amount of time. The control dynamics determines the overshoot of θ and settling time, etc. The inertia load may have high moment of inertia as in a radar antenna, or it may be a small inertial load such as a precision instrument. A simple model of a motor driving an inertial load [93] is considered for the demonstration of the proposed real-time checking mechanism.

Under ideal circumstances, the torque τ developed at the shaft of an armature controlled motor is proportional to the armature current i (since the field current i_f is kept constant) and the induced emf/back-emf v is proportional to the angular speed of rotation ω.

$$\tau = K_1 i \tag{2.10}$$

$$v = K_2 \omega \tag{2.11}$$

The input voltage u is related to the induced emf v as

$$u - v = Ri \tag{2.12}$$

where R is the electrical resistance of the motor armature, (the $L\frac{di}{dt}$ term for armature inductance L is ignored, since it is common practice to assume the time constant of the electrical circuit to be small compared to the time constant of the load dynamics in case of

position control applications). The torque τ is also given as

$$\tau = J\frac{d\omega}{dt} \tag{2.13}$$

where, J is the inertia of the load. Using (2.10),

$$J\frac{d\omega}{dt} = K_1 i = \frac{K_1}{R}(u - v) \tag{2.14}$$

Using (2.11)

$$J\frac{d\omega}{dt} = \frac{K_1}{R}u - \frac{K_1 K_2}{R}\omega$$

or

$$\frac{d\omega}{dt} = \frac{K_1}{JR}u - \frac{K_1 K_2}{JR}\omega \tag{2.15}$$

For control of position θ of the shaft,

$$\frac{d\theta}{dt} = \omega \tag{2.16}$$

Combining (2.15) and (2.16), the state space representation of the dynamics of the motor driving inertia load is given by, as shown in Figure 2.3,

$$\begin{bmatrix} \dot{\theta} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & \alpha \end{bmatrix} \begin{bmatrix} \theta \\ \omega \end{bmatrix} + \begin{bmatrix} 0 \\ \beta \end{bmatrix} u \tag{2.17}$$

where $\alpha = -\frac{K_1 K_2}{JR}$ and $\beta = \frac{K_1}{JR}$.

2.2.1 Proportional Control

Let θ_{ref} be the desired angular position of the inertial load starting with $\theta(0) = 0$. Then if $\theta(t)$ is the controlled variable, $e_\theta(t) = \theta_{ref} - \theta(t)$ is the error in the controlled variable

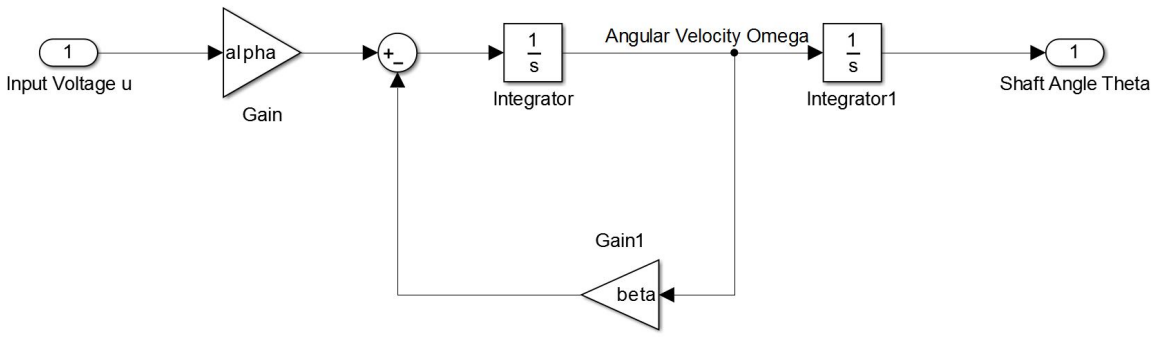


Figure 2.3: Block Diagram for dynamics of the motor-load system

and the objective of the control procedure is to achieve $e_\theta(t) = 0$. The control law is also modified to incorporate ω [93] in case of position control systems, since, unlike speed control applications, the steady state speed should be ideally zero. Thus the control law for negative feedback can be written as

$$u(t) = g_1 e_\theta(t) - g_2 \omega(t) \tag{2.18}$$

where g_1 and g_2 are the proportional gains that are defined according to the pole placement formula proposed by Bass-Gura [94] as:

$$g_1 = \frac{\bar{a_2}}{\beta} \text{ and } g_2 = \frac{\bar{a_1} - \alpha}{\beta} \tag{2.19}$$

where $\bar{a_1}$ and $\bar{a_2}$ are the coefficients of the desired characteristic polynomial for shaping the dynamic response. In practice, the states θ and ω are sensed by an angular potentiometer and a tachometer respectively. Thus the state space representation of the proportional control system can be written as:

$$\begin{bmatrix} \dot{\theta} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\beta g_1 & \alpha - \beta g_2 \end{bmatrix} \begin{bmatrix} \theta \\ \omega \end{bmatrix} + \begin{bmatrix} 0 \\ \beta g_1 \end{bmatrix} \theta_{ref} \tag{2.20}$$

$$\theta = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} \theta \\ \omega \end{bmatrix} \tag{2.21}$$

2.2.2 PID Control

In contrast to the proportional controller described above, a PID controller has a control law of the form

$$u_{PID}(t) = Pe_\theta(t) + I \int e_\theta(t)dt + D\frac{de_\theta(t)}{dt} \tag{2.22}$$

where P, I and D are the proportional, integral and derivative gains respectively. However, a pure derivative term is never used in practice as it allows noise to propagate through the controller and affect its closed-loop performance. Hence the derivative term is used in conjunction with a low-pass filter as shown in Figure 2.4 where the bandwidth of the filter is carefully selected to trade off the speed of the controller response against its noise performance. Thus the transfer function of a practical parallel PID controller is given by

$$C_{par}(s) = P + I\frac{1}{s} + D\frac{NS}{s+N} \tag{2.23}$$

where, the filter coefficient N sets the location of the pole of the derivative filter.

As discussed earlier in (2.18), $u(t) = g_1 e_\theta(t) - g_2\omega(t)$ describes feedback control for a proportional controller. For simplicity, the gain factor g_1 for the error e_θ is replaced by a PID controller but proportional control is retained with gain g_2 along the feedback path of ω. The state space representation of the controller is given in observable canonical form [93], since the error detection block operates directly on the controller states:

$$\begin{bmatrix} \dot{u}_{PID} - (DN+P)\dot{e}_\theta \\ (IN)e_\theta \end{bmatrix} = \begin{bmatrix} -N & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} u_{PID} - (DN+P)e_\theta \\ (IN)\int e_\theta \end{bmatrix} + \begin{bmatrix} I - DN^2 \\ IN \end{bmatrix} e_\theta \tag{2.24}$$

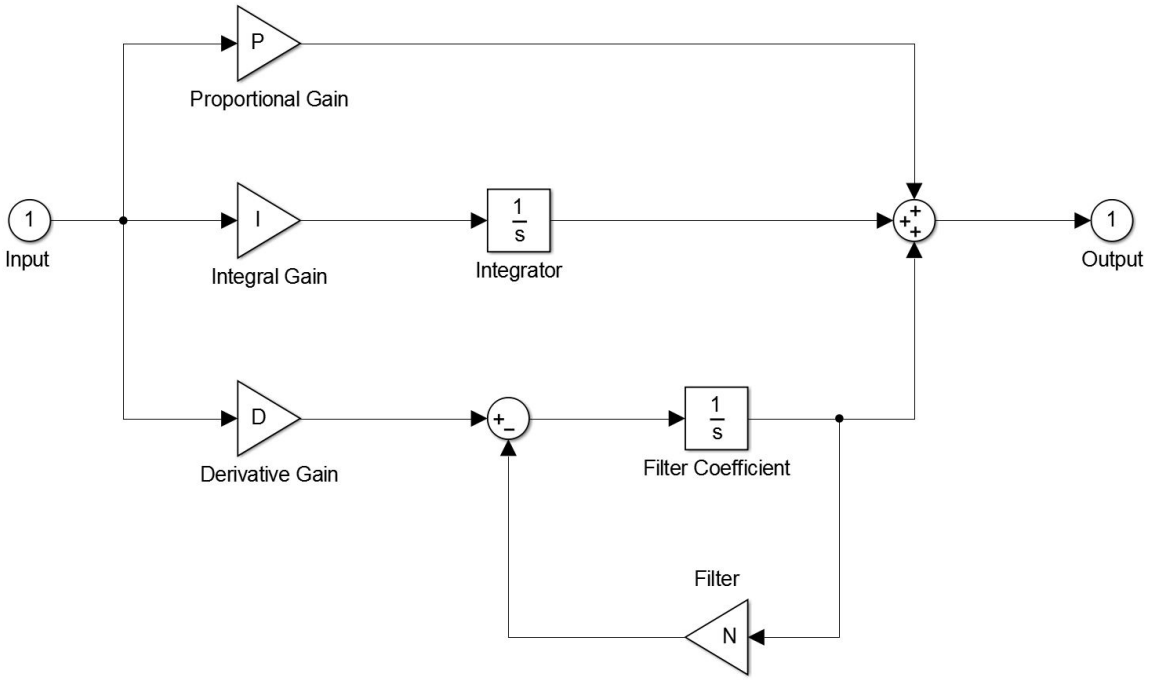


Figure 2.4: Practical implementation of a PID controller

$$u_{PID} = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} u_{PID} - (DN + P)e_\theta \\ (IN) \int e_\theta \end{bmatrix} + (DN + P)e_\theta \tag{2.25}$$

Thus the net control law is $u = u_{PID} - g_2\omega$. Incorporating this into (2.17) and writing the entire system equation as in (2.1),

$$\begin{bmatrix} \dot{\theta} \\ \dot{\omega} \\ \dot{u}_{PID} - (DN + P)\dot{e}_\theta \\ (IN)e_\theta \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -\beta(DN + P) & \alpha - \beta g_2 & \beta & 0 \\ -(I - DN^2) & 0 & -N & 1 \\ -IN & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \theta \\ \omega \\ u_{PID} - (DN + P)e_\theta \\ (IN) \int e_\theta \end{bmatrix} + \begin{bmatrix} 0 \\ \beta(DN + P) \\ (I - DN^2) \\ IN \end{bmatrix} \theta_{ref} \tag{2.26}$$

$$\theta = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \theta \\ \omega \\ u_{PID} - (DN + P)e_\theta \\ (IN) \int e_\theta \end{bmatrix} \tag{2.27}$$

This is of the similar format as in (2.1) by noting that

$$\dot{V}_{PID} = M_{PID} V_{PID} + N_{PID} w_{PID} \tag{2.28}$$

2.3 Proposed Real-time Checking Mechanism

The checking circuit for the two control systems described in Section 2.2 can be constructed according to the theory described in (2.2). The dimension of the coding vector CV is equal to the number of states in the system state space representation.

2.3.1 Proportional Control

The coding vector as described in Section 2.1 earlier, is given as $CV_P = \begin{bmatrix} \alpha_1 & \alpha_2 \end{bmatrix}$. The index P represents the coefficients for the case of P-control only. The matrices S and T as defined in Section 2.1 are given by

$$S_P = \begin{bmatrix} -\alpha\beta g_1 & \alpha_1 + \alpha_2(\alpha - \beta g_2) \end{bmatrix} \tag{2.29}$$

$$T_P = \begin{bmatrix} \alpha_2 \beta g_1 \end{bmatrix} \tag{2.30}$$

The check variable $c(t)$ is given by $c_P = \alpha_1 \theta + \alpha_2 \omega$ and the checksum error is defined as

$$e_P = c_P - S_P \int \begin{bmatrix} \theta \\ \omega \end{bmatrix} - T_P \int \theta_{ref} \tag{2.31}$$

The schematic of the motor/controller and the checking circuitry is shown in Figure 2.5.

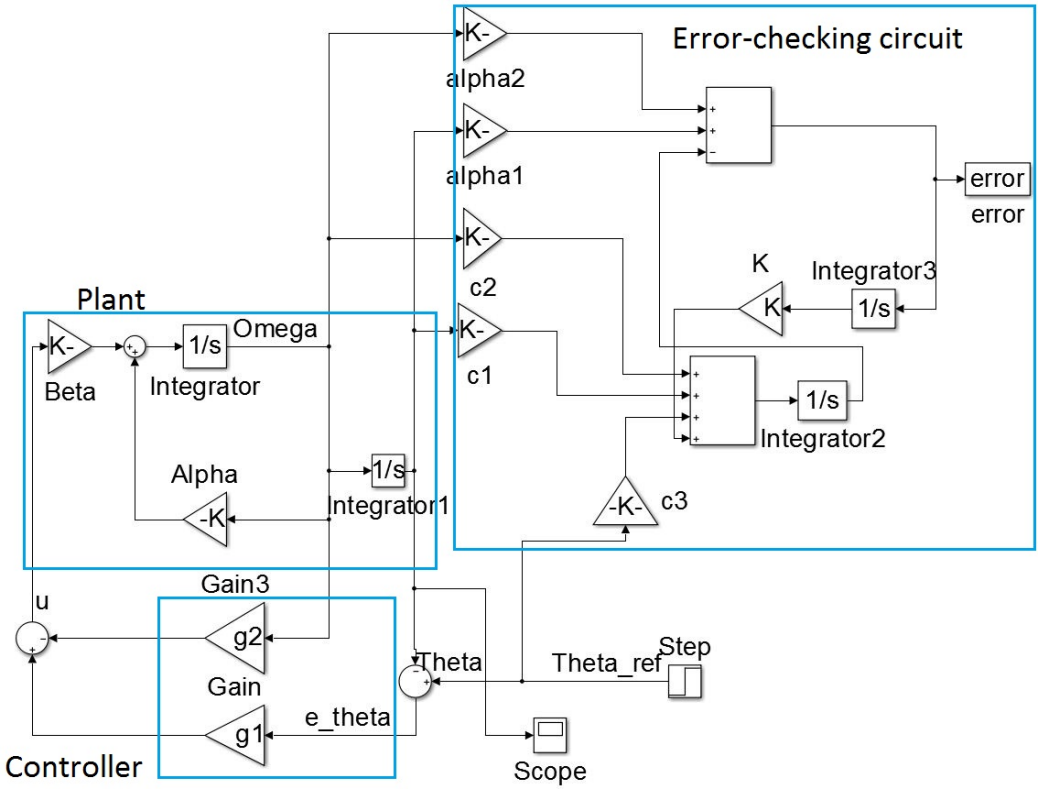


Figure 2.5: Simulink Model for the proposed scheme using P-control

2.3.2 PID Control

For PID control, the coding vector is given as $CV_{PID} = \begin{bmatrix} \alpha_1 & \alpha_2 & \alpha_3 & \alpha_4 \end{bmatrix}$. The matrices S_{PID} and T_{PID} in this case can be calculated similarly as in (2.29)(2.30) and are given by $S_{PID} = CV.M_{PID}$ and $T_{PID} = CV.N_{PID}$. Thus, the check variable in this case can be written as $c_{PID} = \alpha_1\theta + \alpha_2\omega + \alpha_3(u_{PID} - (DN+P)e_\theta) + \alpha_4(IN)\int e_\theta$ and the checksum error is given by:

$$e_{PID} = c_{PID} - S_{PID}\int \begin{bmatrix} \theta \\ \omega \\ u_{PID} - (DN+P)e_\theta \\ (IN)\int e_\theta \end{bmatrix} - T_{PID}\int \theta_{ref} \qquad (2.32)$$

The schematic of the motor/PID controller and the checking circuitry is shown in Figure 2.6.

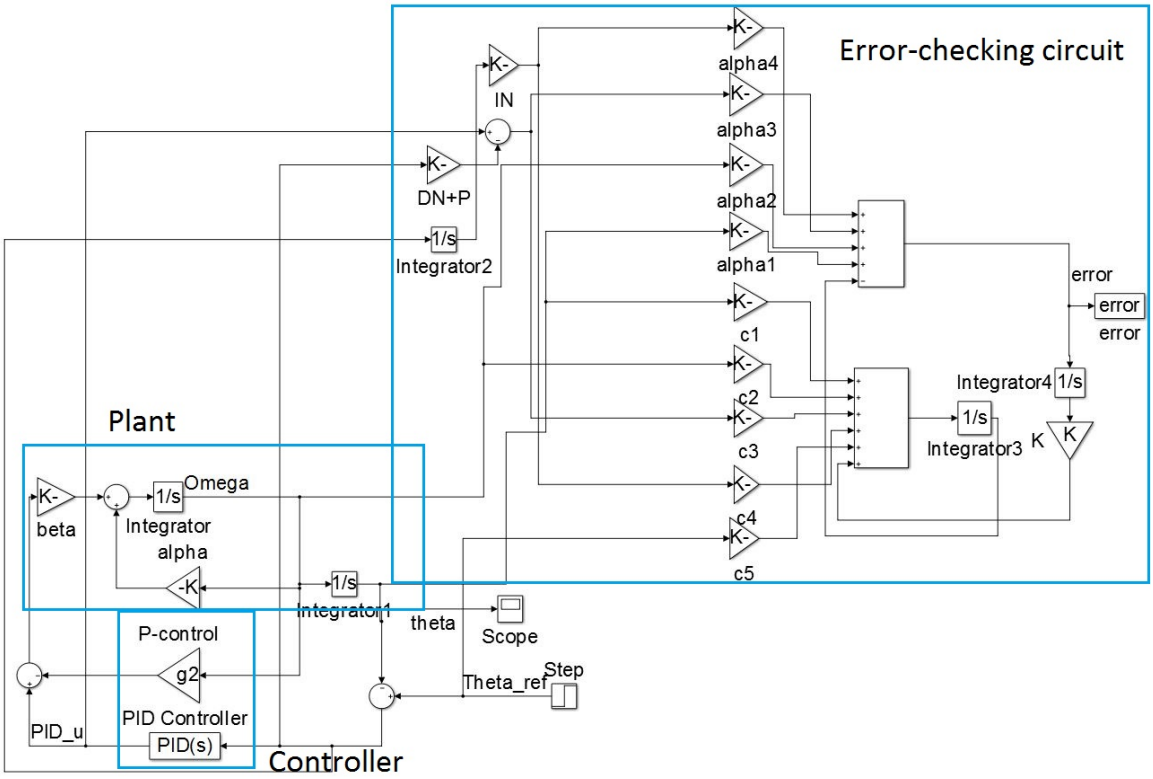


Figure 2.6: Simulink model for the proposed scheme using PID-control

For both proportional and PID control all the weights of the coding vector were chosen to be 1. This is under the assumption that both θ and ω, the system states, are equally important for the performance of the control system. In general, the α_is should be weighted relative to the impact of the state variable $V_i(t)$ on system level performance. The overhead for implementing the checking scheme is very small and consists of an external integrator and summation modules.

2.4 Simulation Results - Detection

2.4.1 Fault Models

The following fault modes (a), (b) and (c) below are considered for both proportional and PID control architectures:

(a) Power Supply Transients: The effects of power supply transients are modeled by representing the torque τ as

$$\tau(t) = K_1(t)i(t) \tag{2.33}$$

where momentary degradation of the produced motor torque is modeled by instantaneous changes in the torque constant $K_1(t)$. In the event that the power shuts off completely for a short interval of time, $K_1(t) = 0$ for the interval of time that power is shut off and returns to its normal value for all other times.

(b) Permanent change in torque constant: The torque-current relationship can be permanently altered under failure by a different torque constant and a DC offset. This is modeled as

$$\tau(t) = K_1^{'}i(t) \tag{2.34}$$

where, $K_1^{'} = (K_1 + \delta K_1), \delta K_1$ being the offset from the rated value of the motor torque constant.

(c) Parameter Variation: Any nominal parameter variations in the plant and controller modules of Figures 2.5 and 2.6 can be modeled as variations in the gain values of the amplifiers in the figures. This models failures in the controller circuitry as well as parametric deviations in the state equations of the servo motor itself.

2.4.2 Simulation Data

For simulation purposes, the following values are chosen for the parameters of the motor: K_1=0.057 N-m/A, K_2=0.052 V-s/rad, J=900 x 10^{-3} kg-m^2. The gain K for the integrator in the feedback path of the error checking circuit is chosen to be K=0.001.

- Proportional Control The gains g_1 and g_2 in (2.19) are selected according to the desired response. The desired characteristic polynomial is selected so that the damping factor is 0.7 and natural frequency is 1 rad/s. The reference input is a unit step volt-

age. The output plots and the checksum error plots are shown in Figures 2.7 and 2.8 respectively.

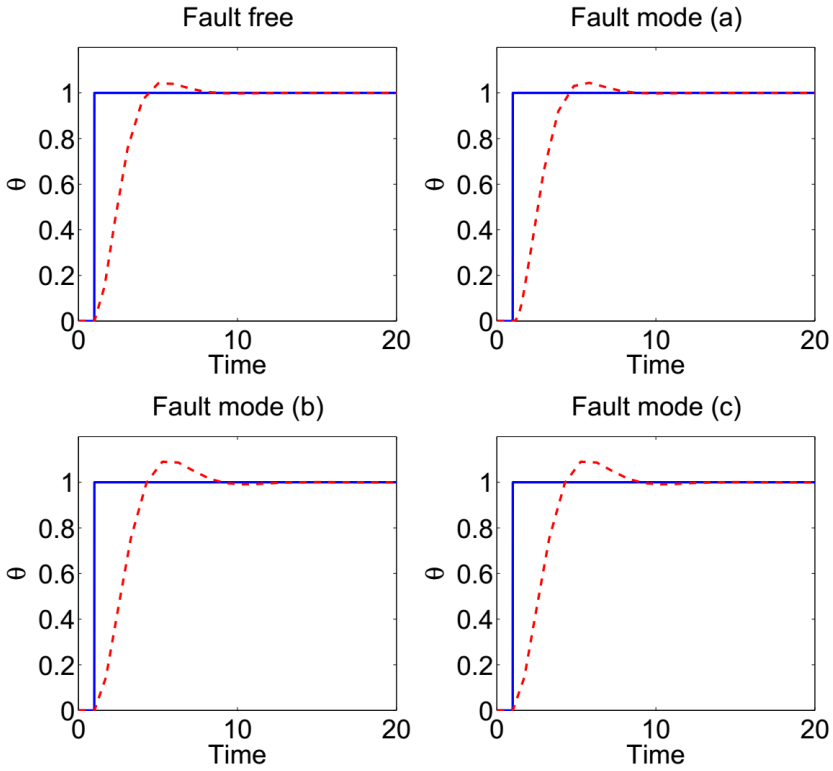


Figure 2.7: Output plots for P-control (solid line: reference input, dotted line : shaft position): a) Nominal case b) Power Supply Transients, c) Torque Constant change d) Controller parameter variation

Note that for the fault free case, the error is zero but for each of the failure modes (b-d), the error signal is non-zero. Also note that the presence of a failure is difficult to deduce from direct examination of the servo motor response as shown in Figure 2.7. Figure 2.9 shows the control input voltage to the motor in the case of a power supply transient, which, if undetected, can lead to motor/controller damage due to the large transients produced. Hence, potential damage to the motor/controller can be averted through the use of the proposed real time checking scheme.

- PID Control - The PID controller parameters are selected for closed-loop stability,

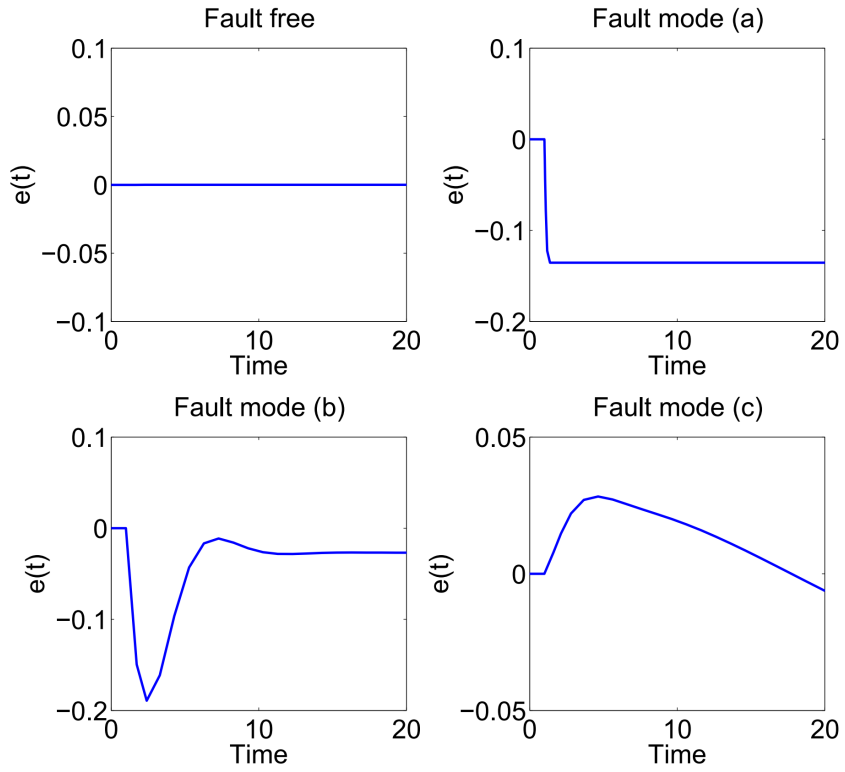


Figure 2.8: Checksum errors for P-control: a) Nominal case (zero error), b) Power supply transient, c) Torque constant variation, d) Controller parameter variation

performance and robustness. Instead of directly using heuristic techniques such as Zeigler-Nichols tuning or Cohen-Coon method [95], controller parameters are selected using loop optimization software based on the open loop response of the linearized model as this method gives consistent results. The PID controller parameters are chosen to be P=20.0032, I= 0.138, D=0.049 and N=90.11. The output plots and the checksum error plots are shown in Figures 2.10 and 2.11, respectively, for the fault-free system and the failure modes (a-c) described earlier. This demonstrates the correctness and efficiency of the proposed real-time checking scheme.

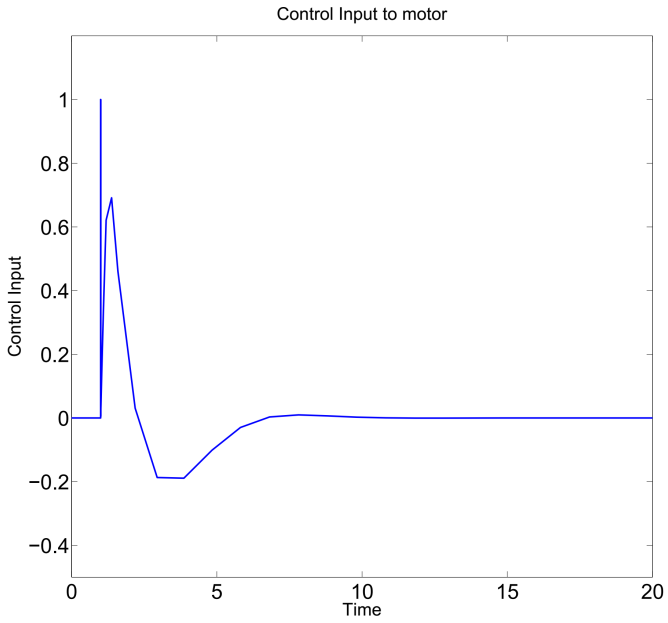


Figure 2.9: Control Input to Motor in case of power supply transient

2.5 Simulation Results - Compensation

During compensation, the error signal is fed back to the control input with a gain of $\rho = 100$. The ability of error signal feedback as a compensation scheme is demonstrated in Figures 2.12 and 2.13 where power supply transients and torque constant shift are considered respectively for the proportional control case. It is clearly seen that the error signal has a high non-zero value when the error signal is not fed back to the control input as compensation whereas, in the case of compensation, the error signal is almost zero demonstrating almost perfect compensation. The small non-zero error signal is explained by the fact that the error compensation is not perfectly accurate. The theory of analog checksums and compensation mentions that accurate compensation should be performed to each state where the affected state feeds back to. However, it can be seen in the signal flow graph of Figure 2.3 that the feedback loop with gain β is inherently within the servo-motor and is not externally accessible or controllable. Hence, the error compensation is not perfect, resulting in the small non-zero error signal even after compensation as shown in Figures

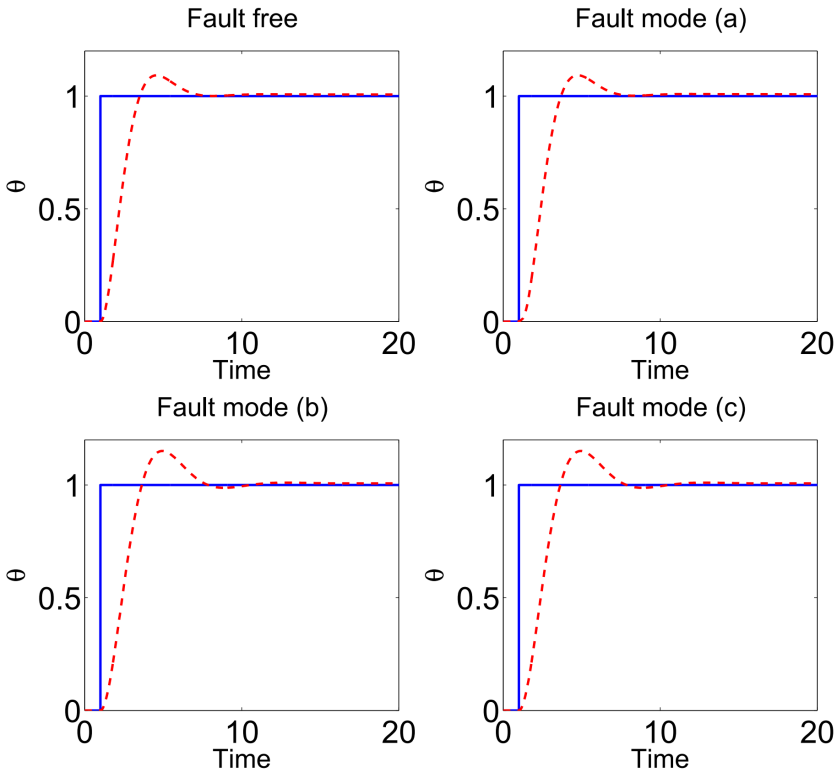


Figure 2.10: Output plots for PID-control (solid line: reference input, dotted line: shaft position): a) Nominal case b) Power Supply Transients, c) Torque Constant change d) Controller parameter variation

2.12 and 2.13.

2.6 Summary

This work illustrates how a real-time monitoring scheme for linear control systems can be proposed with the concept of checksums. This error checking methodology is simple, yet very powerful for detecting transient errors and permanent faults in the circuit in real-time. The proposed approach is demonstrated on a servo-motor control system for the first time and shows excellent failure coverage and detection performance. In addition, it has also been shown [96] how the error signal can be intelligently exploited for failure compensation in real-time with low latency. Two distinct effects of power supply transient

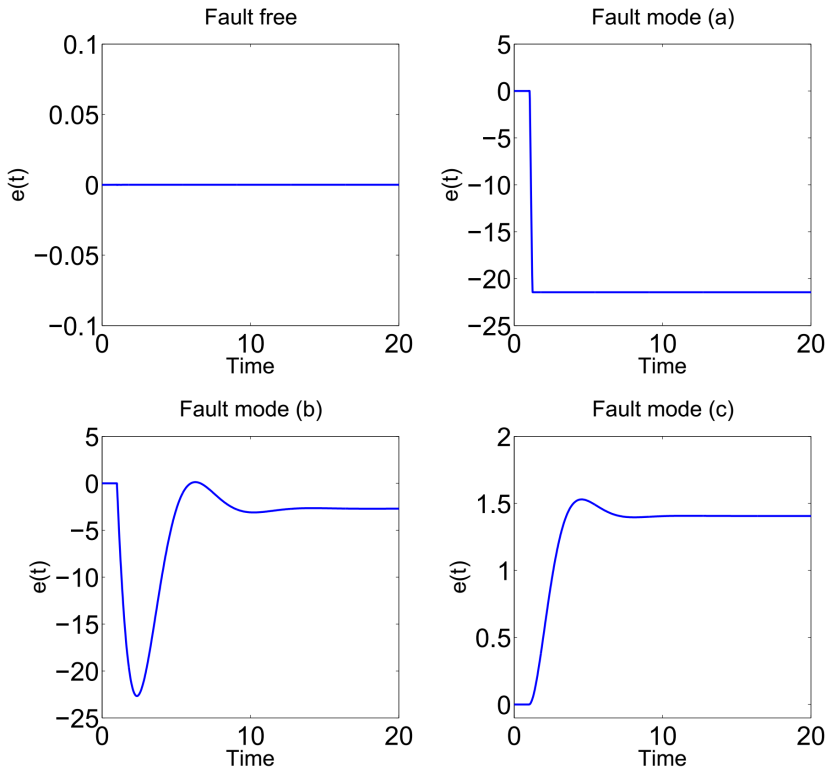


Figure 2.11: Checksum errors for PID-control: a) Nominal case (zero error), b) Power supply transient, c) Torque constant variation, d) Controller parameter variation

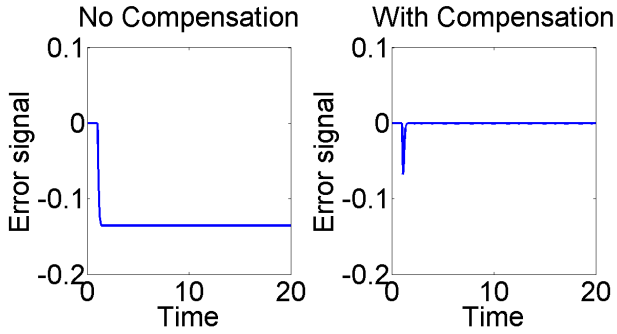


Figure 2.12: Error Signal Plots for Supply Transient Injection: Without and With Compensation

and torque constant shift have been considered and it is seen that the proposed scheme performs near-accurate compensation and excellent correction of induced errors. However, one of the critical requirements for the implementation of correction is the diagnosis of erroneous state. It has been shown in [97] how the linear checksums can directly be used

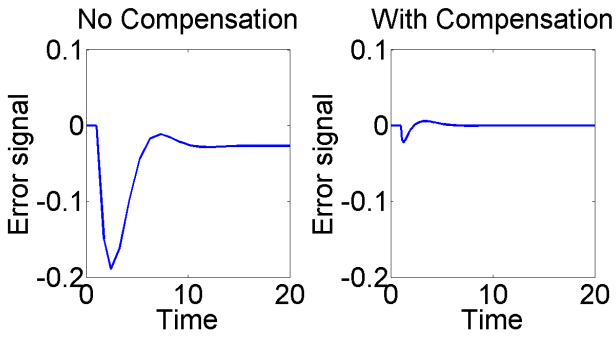


Figure 2.13: Error Signal Plots for Torque Constant Shift: Without and With Compensation

to reconfigure the control law in real-time. In this chapter, it is assumed that accurate knowledge of the erroneous state is available thus enabling us to provide feedback of the error signal into the exact affected state. In the next chapter, extend the error detection and compensation methodologies are extended for analog circuits and two novel diagnosis schemes are introduced along with hardware demonstration.

CHAPTER 3
ERROR CORRECTION IN CIRCUITS

3.1 Real-Time Noise Cancellation in Linear Analog Filters

Transient errors and noise induced by signal coupling, electromagnetic interference and power supply/ground bounce are of major concern in analog circuits since these are difficult to simulate pre-silicon and very difficult to diagnose by current design verification algorithms that are designed mostly for digital timing validation and property checking [98, 99]. In this chapter, the previous methodology is extended for real-time detection and correction of transient errors and induced noise in linear analog circuits. The key assumption is that the noise statistics is stationary and noise injection is localized to within a single state variable of the circuit (integrator and associated summation units). Less than minimum distance checksum codes (distance-2, with low hardware overhear) are used for error correction. The source of the noise and the appropriate correction strategy for noise cancellation is "learned" over a period of real-time operation. The noise cancellation is appropriate, for example, to cancel out localized signal coupling and power/ground bounce and interference effects in deeply scaled and integrated SoCs. Another key benefit of the checksum based error/noise cancellation approach is that it is independent of the kind of noise that is injected, requiring no direct noise modeling or sensing.

3.1.1 State Space System Representation

Redundant States and Error Signal Definition

Consider an n-th order dynamic linear system with p inputs and q outputs. Steady state operation and null initial conditions are assumed. It is well known from the literature

[100]that it admits a state space representation in the frequency domain as,

$$sX = AX + BU \tag{3.1}$$

Vector X represents the n states of the system and vector U corresponds to the vector of p inputs. Matrix A is $n \times n$ while matrix B is $n \times p$. The output is usually written as a linear combination of the states and inputs as $Y = CX + DU$, where Y is the vector of q outputs and C and D are matrices with dimensions $q \times n$ and $q \times p$ respectively.

Let $\alpha = [\alpha_1, \ldots, \alpha_n]$ be a vector of n scalars representing the coding vector. The new redundant state x_r is defined by multiplying the *coding vector* by matrices A and B, therefore expanding the system with an extra row,

$$s\left[\begin{array}{c} X \\ \hline x_r \end{array}\right] = \left[\begin{array}{c|c} A & \mathbf{0}_{n\times 1} \\ \hline \alpha A & 0 \end{array}\right]\left[\begin{array}{c} X \\ \hline x_r \end{array}\right] + \left[\begin{array}{c} B \\ \hline \alpha B \end{array}\right] U \tag{3.2}$$

Here, for the sake of simplicity, the definition of one extra state has been considered, but this can be generalized to any number of redundant states. In such a case, vector α is replaced by a matrix of scalars [73]. This way of defining an extra state allows us to write $sx_r = \alpha AX + \alpha BU = \alpha(AX + BU) = \alpha sX$, or equivalently $x_r = \alpha X$. This proves that state x_r corresponds to a linear combination of the states in system (3.1). Under such conditions, an analog error signal can be defined as the difference of αX and x_r,

$$E = \alpha X - x_r \tag{3.3}$$

This error signal is zero unless the system is perturbed by a transient error. Figure 3.1 depicts the signal flow graph of the state space system represented in Equation (3.2). The generation of the redundant state x_r and the analog error signal E are shown in Figure 3.1. Since the the system defined in Equation (3.2) presents an open loop integrator, the system is marginally unstable. In order to ensure stability, a feedback factor β is applied between

the error signal and the node sx_r. The sign of β should be properly chosen to ensure the added pole remains in the left half s-plane.

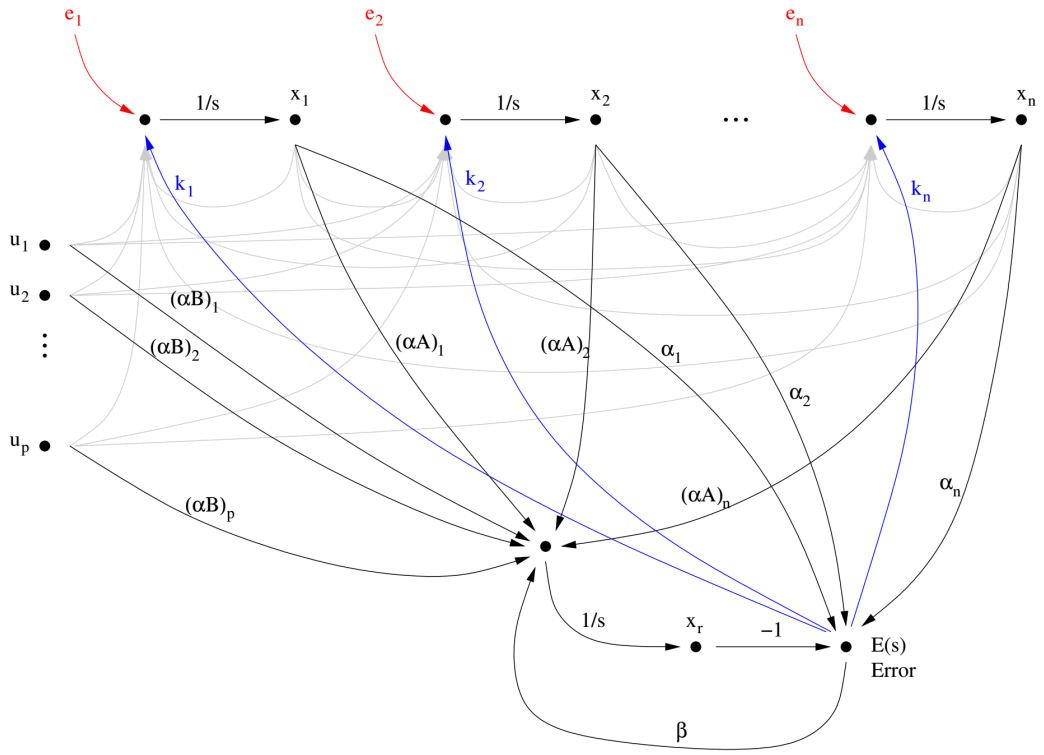


Figure 3.1: Signal flow graph representing a state space system and the generation of the extra state x_r as defined in Equation (3.4). Error signal E is generated by subtracting x_r from αX. Feedback β is added for ensuring stability. Perturbations are indicated in red while error/noise cancellation feedbacks are indicated in blue.

The feedback factor β added for stabilizing the system (3.2) modifies the state equations, consequently, the node sx_r depends on the value of β and the newly created state x_r. The new state space representation of the system becomes,

$$s\left[\begin{array}{c} X \\ \hline x_r \end{array}\right] = \left[\begin{array}{c|c} A & \mathbf{0}_{n\times 1} \\ \hline \alpha(A+\beta I) & -\beta \end{array}\right] \left[\begin{array}{c} X \\ \hline x_r \end{array}\right] + \left[\begin{array}{c} B \\ \hline \alpha B \end{array}\right] U \tag{3.4}$$

where I represents the identity matrix of order n. It is easy to show again that the previous definition of the error signal is still consistent and is zero in the absence of induced errors. From Equation (3.4), it is clear that $sx_r = \alpha(A+\beta I)X - \beta x_r + \alpha BU$. Rearranging terms yields $s(\alpha X - x_r) = -\beta(\alpha X - x_r)$. Recalling the previous definition for the error signal,

$E = \alpha X - x_r$, it immediately follows that $sE = -\beta E$, what necessary implies $E = 0$, under the assumption of null initial conditions and therefore $\alpha X = x_r$.

Feedback for Error/Noise Cancellation

Consider the system defined by Equation (3.1). Let us assume that the i-th state of the system is perturbed by a noise signal $n(t)$. Therefore the system under study is given by,

$$s\widetilde{X} = A\widetilde{X} + BU + \delta_i N \tag{3.5}$$

where N corresponds to the Laplace transform of the perturbing signal $n(t)$ and δ_i is a column vector in which all elements are zero with the exception of the element in position i, which is 1. Vector $\widetilde{X}$ indicates the modified states under the influence of a perturbing signal. In the following it will be shown that feeding back the error signal to the i-th state with a sufficiently large gain cancels the effect of the transient perturbation in the system. Let k be the gain of this feedback, then the system becomes,

$$s\widetilde{X} = A\widetilde{X} + BU + \delta_i N + \delta_i k E \tag{3.6}$$

It is simple to obtain an expression for sE since, by definition, $sE = s\alpha\widetilde{X} - s\widetilde{x}_r$. Therefore, the first term of the difference is $\alpha(A\widetilde{X} + BU + \delta_i N + \delta_i k E)$. The second term can be computed according to Equation (3.4) as $\alpha(A+\beta I)\widetilde{X} - \beta\widetilde{x}_r + \alpha BU$. Equating their difference, some terms cancel out, therefore giving $sE = \alpha_i N + \alpha_i k E - \beta(\alpha\widetilde{X} - \widetilde{x}_r)$, where α_i is the i-th element of the *coding vector* α. Again, the difference $\alpha\widetilde{X} - \widetilde{x}_r$ corresponds to the definition of the error signal, so $sE = \alpha_i N + \alpha_i k E - \beta E$, or equivalently,

$$E = \frac{\alpha_i}{s + \beta - \alpha_i k} N \tag{3.7}$$

The transfer function in Equation (3.7) corresponds to a first order linear system with a single pole $p = -(\beta - \alpha_i k)$. As discussed previously, the sign of β should be positive and

the sign of the feedback factor k should be negative in order to ensure that the system is stable. Feedback factors β and k have a significant impact on system stability, so need to be chosen accurately. Also, their signs depend on the sign of the integrators used to implement the system. Computing the limit of expression (3.7) when $|k| \to \infty$ follows that the error signal tends to zero.

Considering Equation (3.6) and plugging the obtained transfer function for the error signal obtained in Equation (3.7), and after grouping terms, immediately follows,

$$s\widetilde{X} = A\widetilde{X} + BU + \delta_i \underbrace{\frac{s+\beta}{s+\beta-\alpha_i k}}_{\to 0 \text{ if } |k|\to\infty} N \tag{3.8}$$

As can be seen, the noise term tends to zero as $|k| \to \infty$ and therefore the perturbed system $s\widetilde{X} = A\widetilde{X} + BU + \delta_i N + \delta_i kE$ becomes $sX = AX + BU$, the error-free state variable system.

3.1.2 Learning-Assisted Adaptive Error Cancellation

As evident from Equation (3.8), the ability to perform perfect error cancellation is independent of the choice of α_i for large values of the feedback coefficient k. Therefore on only needs to determine which system state is impacted by the injected error and compensate for the error by feeding back the error signal E to the affected state with a large value of k. Since only a single checksum based error detection circuit is implemented, we have in effect, a distance 2 checksum code which is good only for error detection, but not correction. To solve this problem, the proposal is to "learn" which state is impacted by the error through real-time "self-learning" experiments under the assumption that the error statistics is stationary and that the injected errors repeat over time (e.g. due to crosstalk). In practice, the system can be programmed to "learn" in the field over a duration of time (specified by the designer). Once the learning is complete, perfect error cancellation is achieved. Two separate architectures are proposed for diagnosis [101–103] - one implements single error

checksum requiring lower area and power at the cost of high latency of diagnosis while the other implements two error checksums requiring more area and power with the quickest diagnosis along with the enhanced capability of multiple fault compensation. For each of the architecture, two separate learning schemes are proposed.

Single checksum architecture

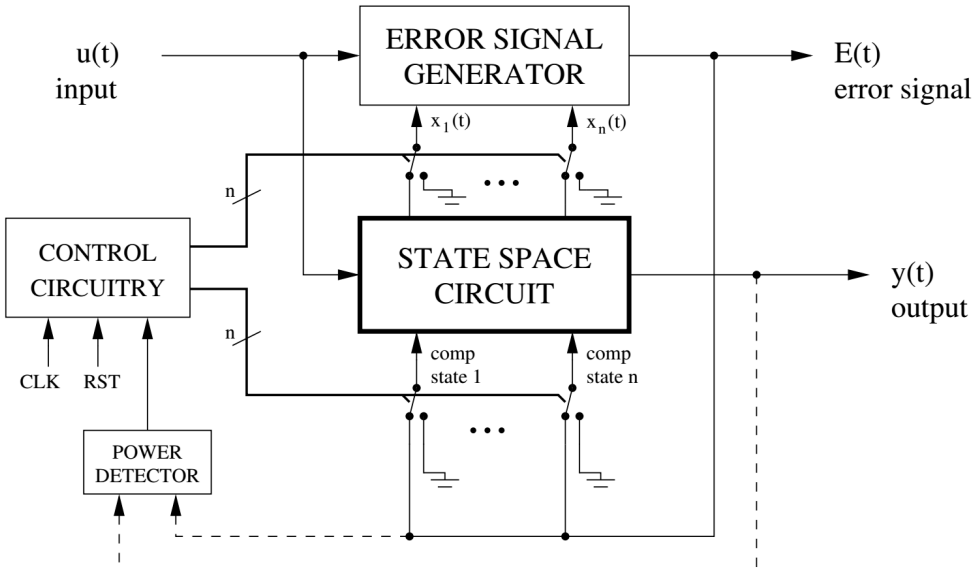


Figure 3.2: Real-time transient error and induced noise cancellation scheme. A hardware-directed search learns where to apply the cancellation signal in order to mitigate the effects of transient error/noise perturbations.

In the first approach (Fig 3.2), the output of the error signal generator (checksum circuit) is selectively fed back to each state in a time-multiplexed sequential manner. The error signal is fed back to the i-th system state (with high gain k) for a predetermined amount of time over which error data is collected (10s of ms, specified by the designer). The control circuitry *actuates a single switch* from the row of switches below the state space circuit of Fig 3.2, that determines the state to which the error signal is fed back. The switches at the top of the state space circuit are all turned "on", connecting the state space circuit to the error signal generator. To diagnose the corrupted state variable of the filter, it must be remembered that the output noise power of the filter is minimized when the

error signal is fed back to the corrupted state (this achieves perfect noise compensation). Conversely, the noise power of the signal is increased when the error signal is fed back to a "good" (not corrupted) state variable. Since the output signal power is the sum of the signal power and the noise power of the output signal $y(t)$, the output power of the signal $y(t)$ of the analog filter (state space circuit in Fig 3.2) is computed for known (DC/sinusoidal) input signal to the filter. The bottom switch of Fig 3.2 for which the output power of the signal $y(t)$ is minimized determines the state to which the error signal must be fed back to achieve perfect transient error/noise compensation. An analog power detector [104] connected to the filter output $y(t)$ is typically used in combination with DC filter input stimulus to facilitate this "learning" procedure. From (3.1), it is known that system output x_n is given as $sx_n = A_nX + B_nU$, where A_n and B_n are the last rows of matrix A and B. It is clear, that if the error is not compensated at the exact state which has been affected, the noise shows up in the output. So, by monitoring the output for the noise power, the faulty state is determined.

In the second approach, the error signal E is monitored *without feeding it back* to the state space circuit (filter). The bottom row of switches of Fig 3.2 are all turned off, disabling error signal feedback. The control circuitry selectively disconnects the i-th state variable input to the error signal generator in a sequential time-multiplexed manner as before, setting the corresponding top row switch to zero (ground). It is easy to show that the error signal is zero in the presence of injected errors/noise only when the corrupted state is disconnected from the error signal generator. Here again a power detector connected to the error signal generator output E is used to detect this condition and determine the corrupted state. Subsequently, after the corrupted state is determined correctly, the correct switch from the bottom row of switches of Fig 3.2 is activated to allow perfect injected error/noise error compensation. From (3.3) and (3.2), it is seen that if the α_i for the faulty state is turned to zero, the contribution of the faulty state to the term αX and the checking state x_r is removed and hence the error signal for the rest of the correct states is zero by

construction. Thus by monitoring the error line and switching the α_is, the faulty state is detected.

Double checksum architecture

It is assumed that for the linear state space system of Fig 3.3, two analog checksums are implemented with error signals ϵ with coding vector α and E with coding vector β.

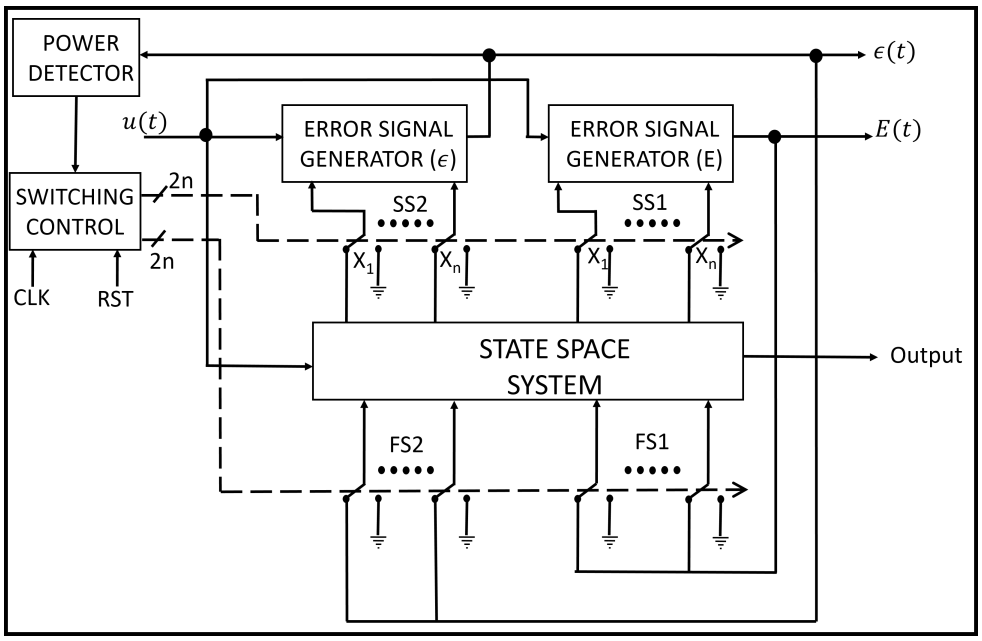


Figure 3.3: Real-time Fault Detection and Compensation scheme

Single Fault Compensation: Two self-learning schemes are proposed for error detection and compensation. The top row switches marked $SS1$ and $SS2$ control the contribution of the individual states to the error signals E and ϵ (by selecting the value of αs and βs to zero or non-zero). The bottom row switches marked $FS1$ and $FS2$ control the feedback of the error signals E and ϵ to the corresponding states.

In the first learning approach (Fig 3.3), the first error signal E is selectively fed back to each system state in a time-multiplexed sequential manner. The control system selects the state to which the error signal E is fed back by actuating a single switch from the row of switches marked $FS1$ in Figure 3.3. Switches $FS2$ are all turned off for single fault

compensation. The corrupted state variable of the filter is diagnosed by an analog power detector monitoring the second error generator output ϵ. The state whose feedback switch connection gives the lowest power for ϵ is determined as the corrupted state. Once the faulty state is diagnosed, the corresponding feedback switch from $FS1$ is connected to the error signal E by the control circuit and all other switches from $FS1$ are disconnected, thereby providing precise compensation to the faulty state. However, after compensation, if one of the error signals is zero whereas the other one is non-zero, then the state space circuit is not faulty and no correction is necessary.

In the second learning approach, the bottom row of switches $FS1$ and $FS2$ are all turned off providing no feedback compensation during the learning phase and the error signal ϵ is monitored. The control system selectively disconnects the j-th switch from the set $SS1$ and $SS2$, thereby disconnecting the j-th state input to the error signal E and ϵ in a similar time-multiplexed manner. The error signals are zero only when the faulty state is disconnected from the error generators. Thus, the faulty state is diagnosed and the corresponding feedback switch from $FS1$ is "turned on" by the control circuitry for precise compensation to the corrupted state.

Dual Fault Correction: The algorithm for dual fault compensation is presented here. Let us assume that the two faulty states are x_k and x_l.

Detection - All the feedback switches $FS1$ and $FS2$ are disconnected, thus providing no compensation. All the switches in $SS1$ are disconnected since only error ϵ is monitored. The control circuitry selectively disconnects each of the switches from the set $SS2$. Disconnection of any one good state implies contribution of the two faulty states to the error line which shows a fixed non-zero value. However, if one of the two bad states is disconnected, the error signal changes.

Compensation - Once the learning is over, the control circuitry disconnects k-th switch from the set $SS1$, thereby zeroing out the contribution of error from x_k and feeds back the error E to the state x_l by connecting the l-th switch from the set $FS1$. Similarly, for the

other error signal generator ϵ, the l-th switch is disconnected from the set $SS2$ and connects the k-th switch from the set $FS2$. Perfect compensation is achieved since the error signals E and ϵ contain only the faulty signals from state W_l and W_k respectively.

3.1.3 Case Study: Band-Pass Butterworth Filter

Band-Pass Butterworth Filter State Space Representation

In order to demonstrate the viability of the proposal, the method has been applied to a linear analog filter. A continuous time 6th order band-pass Butterworth filter [105] can be represented with the following transfer function,

$$H(s) = \frac{-\frac{s^3}{\omega_0^3}}{\frac{s^6}{\omega_0^6} + 2\frac{s^5}{\omega_0^5} + 5\frac{s^4}{\omega_0^4} + 5\frac{s^3}{\omega_0^3} + 5\frac{s^2}{\omega_0^2} + 2\frac{s}{\omega_0} + 1} \tag{3.9}$$

The filter has been tuned at $f_0 = 106.1$ MHz, and $Q = 1$, therefore presenting a bandwidth of BW $= 106.1$ MHz. The Butterworth filter can be described by the signal flow graph depicted in Fig. 3.4. Some weights have been changed in sign to allow the use of inverting integrators thus simplifying its hardware implementation.

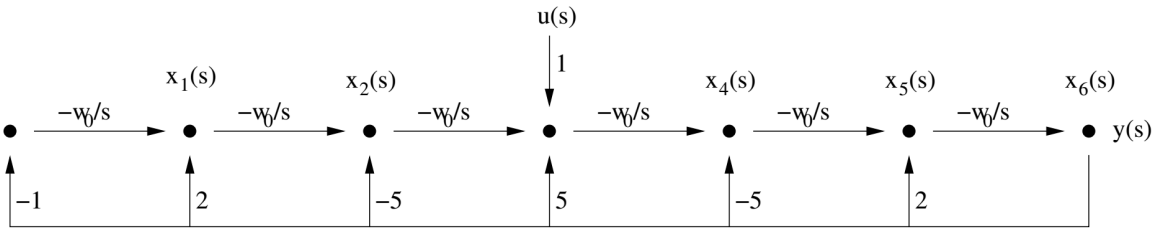


Figure 3.4: Signal flow graph of the 6th order band-pass Butterworth filter. Some feedback weights have been changed in sign to allow the use of inverting integrators therefore facilitating its hardware implementation.

According to the signal flow graph depicted in Figure 3.4, the state space representation of the Butterworth filter can be easily derived. Its output, $y(t)$, corresponds to state $x_6(t)$

as the diagram sketches. The state space representation is,

$$-\frac{s}{\omega_0}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 & 0 & 2 \\ 0 & 1 & 0 & 0 & 0 & -5 \\ 0 & 0 & 1 & 0 & 0 & 5 \\ 0 & 0 & 0 & 1 & 0 & -5 \\ 0 & 0 & 0 & 0 & 1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} u(s) \tag{3.10}$$

In Figure 3.5, the hardware implementation of the 6th order band-pass Butterworth filter using operational amplifiers and passive components can be observed. Second order Sallen-Key stages could have been used as well, but the chosen topology targets to show up the capabilities of the method on its simplest state space form. In the schematic, injected perturbations are denoted as $e_1(t), \ldots, e_6(t)$.

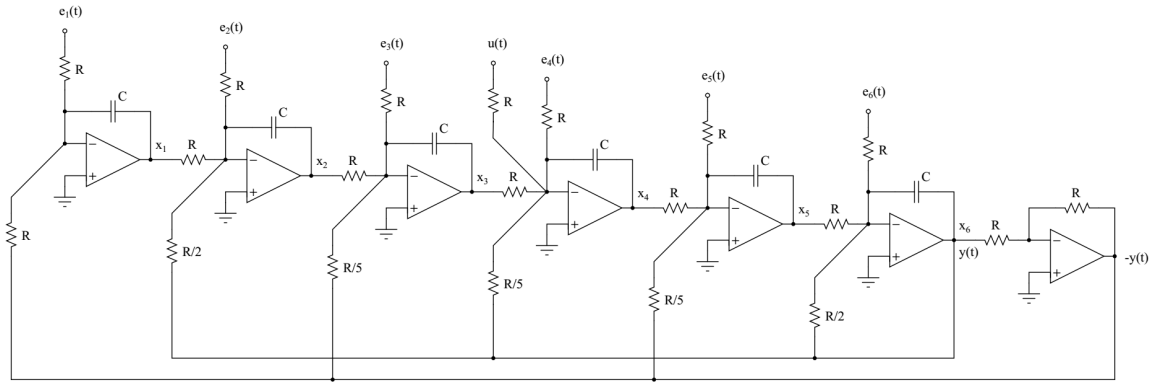


Figure 3.5: Schematic of the 6th order band-pass Butterworth filter whose signal flow graph is depicted in Figure 3.4. The filter has been tuned at $\omega_0 = 100$ MHz and $Q = 1$. States variables correspond to integrator's outputs, $x_1, \ldots, x_n$. Inputs $e_1(t), \ldots, e_6(t)$ correspond to the injected perturbations upsetting the states of the filter.

Extended System for Error/Noise Cancellation

According to the theory developed in Section 3.1.1, the state space representation of the Butterworth filter is expanded with a redundant checksum state variable. Let us select $\alpha = [1, 1, 1, 1, 1, 1]$ as the *coding vector* and a stabilizing feedback factor of $\beta = -1$.

Note that since inverting integrators are being used the sign of β must be negative to ensure stability. According to this selection, the resulting signal flow graph for generating the redundant state x_r and analog error signal E can be observed in Figure 3.6.

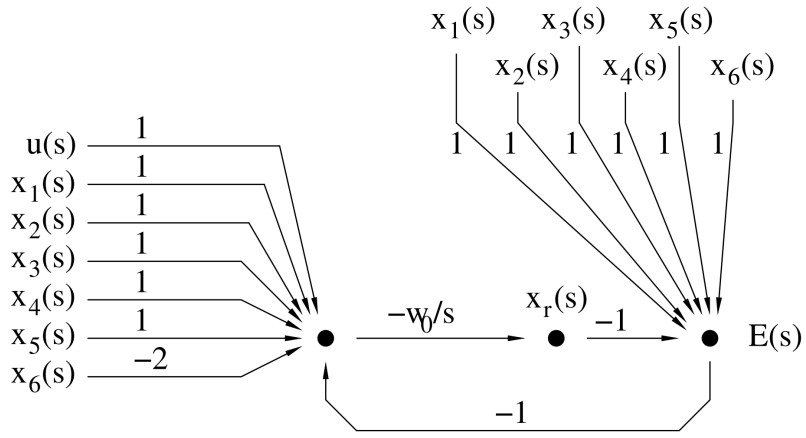


Figure 3.6: Signal flow graph of the analog error signal generator for the 6th order Butterworth filter shown in Equation (3.9). The used *coding vector* is $\alpha = [1, 1, 1, 1, 1, 1, 1]$.

The resulting state space representation of the system after extending it with the redundant state x_r can be computed using Equation (3.4). This way, $A_{6\times6}$ matrix transforms into $\hat{A}_{7\times7}$ and matrix $B_{6\times1}$ transforms into $\hat{B}_{7\times1}$ as indicated below,

$$\hat{A} = -\omega_0 \left[\begin{array}{cccccc|c} 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 1 & 0 & 0 & 0 & -5 & 0 \\ 0 & 0 & 1 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 1 & 0 & -5 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & -3 & 1 \end{array}\right] \quad \hat{B} = -\omega_0 \left[\begin{array}{c} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ \hline 1 \end{array}\right] \tag{3.11}$$

As explained before, transient errors and induced noise cancellation is achieved by feeding back the generated error signal. This feedback is achieved by connecting an appropriate value resistor between the error signal and the virtual ground of the operational amplifier that is affected by the perturbation. The corresponding resistor value determines the k_i feedback for the i-th state. Figure 3.7 shows an HSPICE transient simulation of the

circuits shown in Figure 3.5 when a sine wave is applied to the input. After injection of the perturbing signal as shown in Figure 3.7, the filter response gets corrupted as shown in Figure 3.8. However, the compensated filter response, after the error is fed back with gain $k_3 = 10$, again returns back to the nominal value as shown in Figure 3.9. The error signal, with and without compensation is shown in Figure 3.9. It is seen that without compensation, the error signal resembles the inverted perturbation whereas with compensation, the error signal returns back to almost zero.

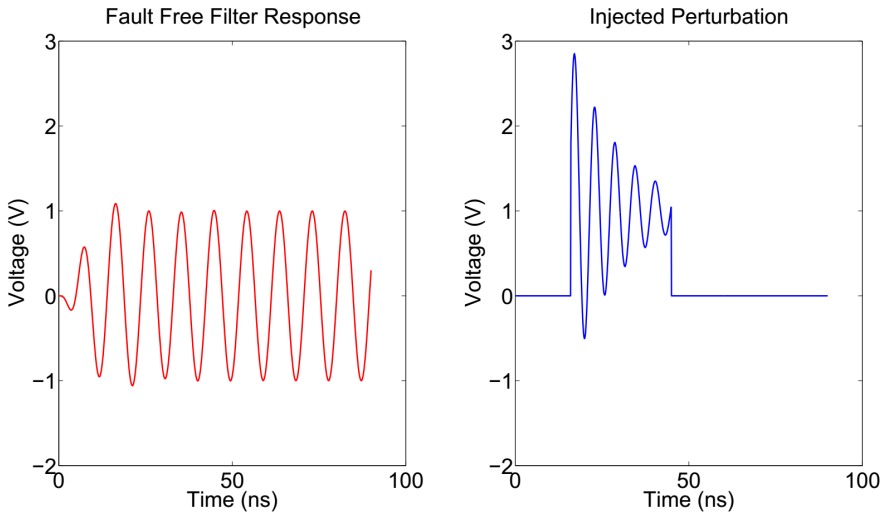


Figure 3.7: Transient simulation of the Butterworth filter showing the fault free filter response and the injected perturbation in state x_3

The effectiveness of the compensation with the frequency and feedback factor k_3 is shown in Figure 3.10. The SNR of output is computed as the ratio of the output's power due to a unitary input sine wave and the output's power due to a unitary sine wave perturbation applied to $e_3(t)$. Figure 3.10 shows the obtained results of the SNR of the filter output for perturbations in state x_3 (similar results are obtained in other states). It is seen that the SNR values in the pass-band approximately correspond to the applied gain in the compensation feedback.

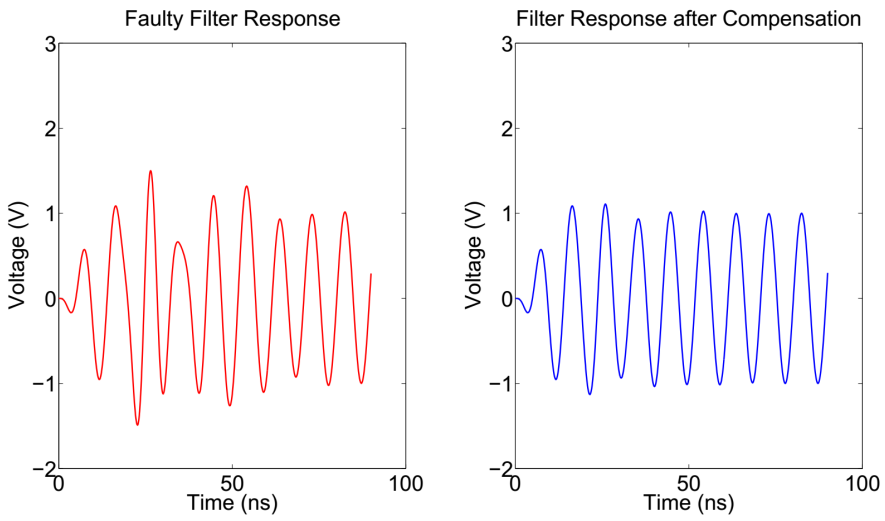


Figure 3.8: Corrupted Filter Response under injected Perturbation and Compensated Filter Response after feedback of the error to state x_3

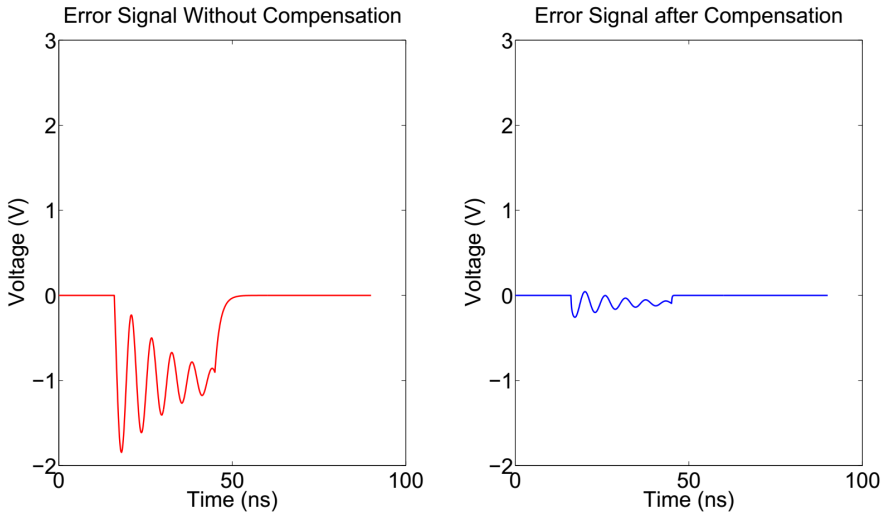


Figure 3.9: Without compensation, the error signal has high non-zero value, whereas after compensation the erro signal almost goes to zero

3.1.4 Experimental Results

In order to validate the proposal a PCB is being built to demonstrate the idea. However, as proof of concept, an experimental test-bench has been built using commercial off-the-shelf components, operating at a lower frequency. Figure 3.11 corresponds to a picture of the constructed test-bench in solderless breadboard using operational amplifiers and passive components. It is important to note the low area overhead with respect to the 6th order

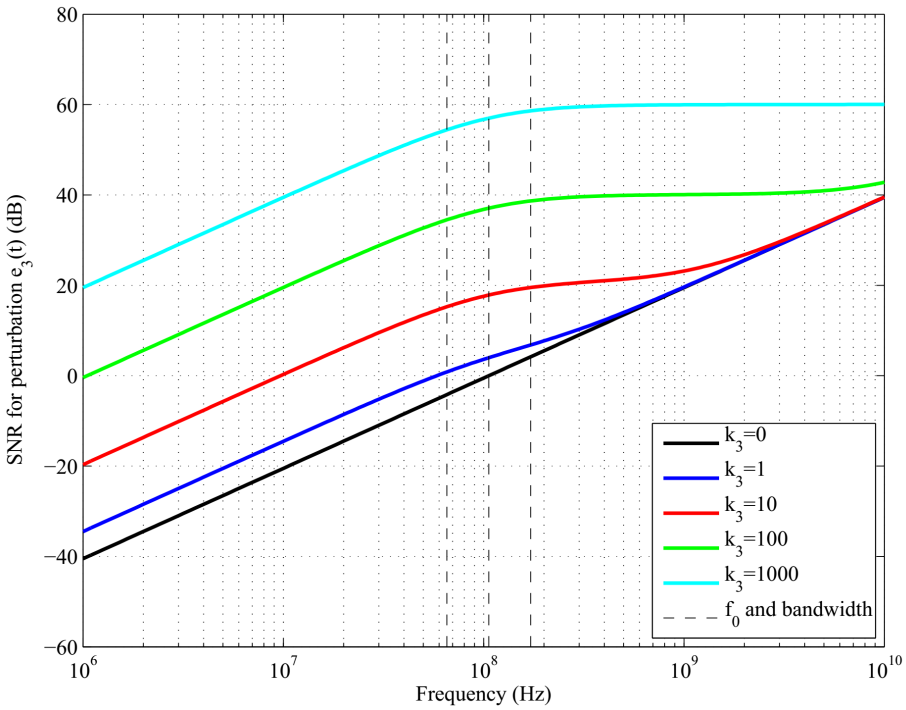


Figure 3.10: Simulation of SNR values at the output of the Butterworth filter for perturbation $e_3(t)$ as a function of the applied compensation gain k_3. Similar values are obtained for perturbations affecting the other states.

band-pass Butterworth filter. The filter uses 7 operational amplifiers as seen in Figure 3.5 while the error generation circuit can be implemented with only 2 op amps. Further this overhead remains constant irrespective of the size of the filter being checked. The overhead can be shown to be $\mathcal{O}(1/n)$, with n being the number of states in the system.

Figure 3.12 shows the output of the filter when a sine wave is applied to its input and a perturbing pulse is applied to $e_3(t)$ with no compensation. Sine wave is 1 V peak-to-peak and about 88 kHz. The perturbing signal is 2 V peak-to-peak and has a high frequency component of about 200 kHz. As can be observed, the output signal gets considerably affected by such perturbation.

On the contrary, consider the oscilloscope capture depicted in Figure 3.13 where an error/noise compensation with gain 10 has been applied. It can be seen how the perturbation effects in filter's output have been strongly mitigated, therefore assessing the viability of the error/noise compensation strategy. Experimental results match those encountered using

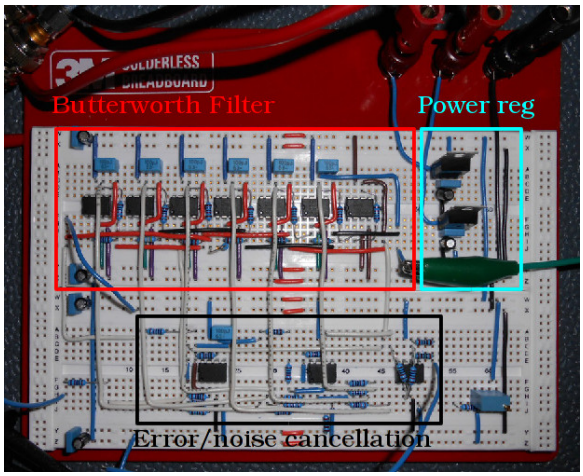


Figure 3.11: Experimental test-bench built using commercial off-the-shelf components. Note the area requirements for the error/noise cancellation circuit and the 6th order Butterworth filter.

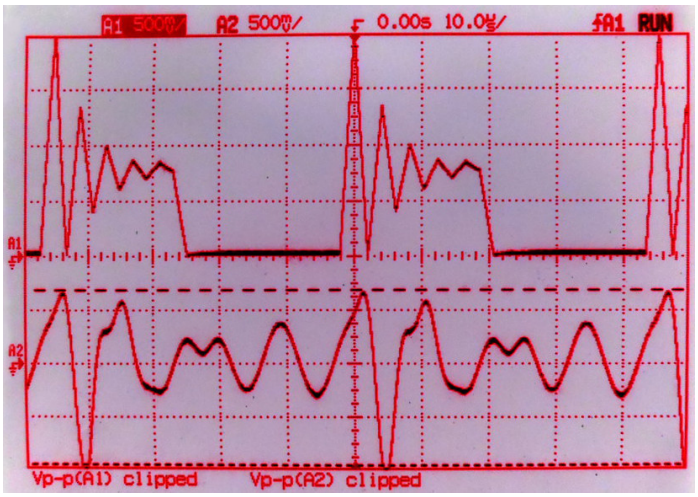


Figure 3.12: Oscilloscope capture when a sine wave is applied to the input and a perturbing noise signal is applied to $e_3(t)$ and no error/noise cancellation is applied. As can be seen, the output of the filter is greatly affected.

HSPICE simulations.

3.1.5 Summary

In this work, the concept of analog checksums proposed in the previous chapter has been extended for linear analog systems for transient error detection and active noise cancellation in analog circuits. The introduced novel approach is based on the use of state space

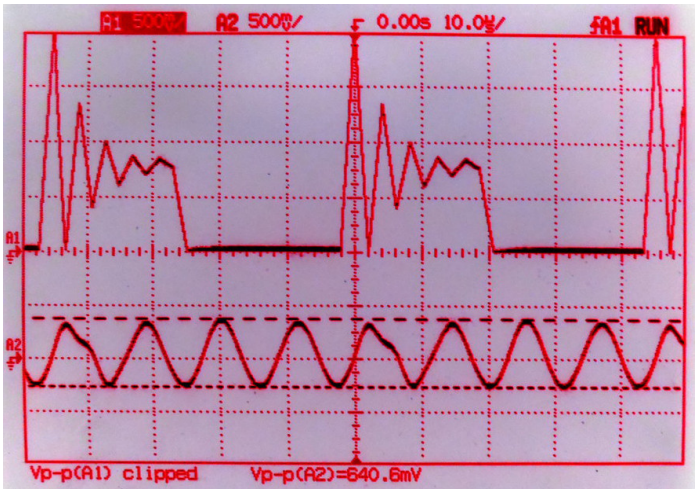


Figure 3.13: Same situation as in Figure 3.12 but feeding back the error signal with gain $k_3 = 10$. It can observed how the compensation strongly mitigates the effects of the perturbation in filter's output.

representations of analog filters and is a significant advancement over prior research that addressed only hard parametric deviations. A key innovation has been the use of less than minimum distance checksum codes for error detection and correction using real-time learning of the likely source of transient errors and noise within the analog circuit. By running a simple hardware-directed search algorithm, the circuit is able to "learn" how best to compensate for the signal disturbances with low overhead under the assumption that the source of the injected errors/noise and the error/noise statistics is stationary over time. Successful simulations and preliminary experimental results conclusively demonstrate almost complete compensation of injected noise, therefore validating the proposal. Next it will be discussed how these methodologies can be further developed for concurrent error detection in nonlinear digital filters.

3.2 Concurrent Error Detection in Nonlinear Digital Filters

3.2.1 Introduction

Till now, the focus has mostly been on error resilience of linear control systems and linear circuit operations. However, there is an increasing need to address the same for nonlin-

ear systems used in emerging applications for sensing and control. Due to the increase in complexity of deep sub-micron high speed digital circuits, they have become more susceptible to soft errors[106–108]. These errors have become more dominant in scaled digital circuits due to feature-size reduction, almost near-doubling of operating frequency and supply voltage/threshold voltage scaling [66, 109]. With each technology generation, the node capacitance values further scale down, thus allowing soft errors to cause erroneous digital logic transitions at these nodes. Such intermittent and permanent failures are of particular concern in digital signal processing, control and communication applications. Lower supply voltage further increases the susceptibility of these circuits to external noise sources due to reduced noise margin. In systems with concurrently operating on-chip digital logic, analog/RF circuitry and mixed signal blocks such as data converters, noise can get coupled through the substrate as well as the power supply and ground planes, causing periodic logic upsets. Present architectural trends with shorter pipelines with reduced slack and higher clock rates make circuits more vulnerable to soft errors.

Of increasing concern, in this context, are nonlinear digital filters that are extensively and routinely used in signal processing, communication and control applications where reliability and dependability are critical issues. Traditional methods for error detection and correction rely on hardware duplication/triplication [63–65, 110]. Error detection in non-linear systems generally involves partitioning of the circuit into linear and nonlinear components. Checksum codes are applied to the linear components for error detection whereas hardware and/or software redundancy is applied for error detection in the nonlinear modules. However, the high overhead in terms of power and area associated with these methods makes them relatively expensive in general applications. A more elegant error detection technique for non-linear systems was proposed in [71] based on *time-freeze linearization* which models a nonlinear digital filter using a time-varying linearized representation. The checksum circuit generates a time-varying checksum code for each single time frame by *freezing* the system dynamics between two adjacent time frames and lin-

earizing the circuit behavior between the two frames. However, the method can result in duplication of all the nonlinear functions in the worst case and further requires that the wordlength precision of the checking circuitry be the same as that of the circuit under test (CUT). This incurs higher cost in terms of both complexity, area and power.

In this chapter, a low cost error detection technique for non-linear digital filters using *linearized checksum codes* and linear predictive coding (LPC) algorithms is presented. First, a least squares linear fit to the nonlinear system is derived and checksum codes are applied to the derived best-fit linear model. When the circuit non-linearities are not excited by the input stimulus (such as for small-signal inputs), the checksum error is ideally zero. In the absence of soft errors, for large input signals, the checksum error carries the non-zero time-varying values which are proportional to the degree of circuit nonlinearity excitation by the stimulus applied. Linear predictive codes are applied to this time-varying error signal to detect soft errors using forward error prediction.

3.2.2 Nonlinear Digital Circuits: Brief Discussion

A generic data flow graph of a nonlinear digital circuit is indicated in Figure 3.14. The nonlinear state variable system implements a nonlinear transformation of the the elements of input vector $\mathbf{u}(t)$ and previous system states $\mathbf{s}(t)$. For example, a possible nonlinear function may be $s(i)^k$ or a product of integer powers of input elements. A general representation of a linear time-varying state-space system (a particular class of nonlinear systems) will be given as,

$$\begin{aligned} \mathbf{s}(t+1) &= \mathbf{A}(t)\mathbf{s}(t) + \mathbf{B}(t)\mathbf{u}(t) \\ \mathbf{y}(t+1) &= \mathbf{C}(t)\mathbf{s}(t) + \mathbf{D}(t)\mathbf{u}(t) \end{aligned} \tag{3.12}$$

where all the matrices contain non-linear combinations of system states and inputs and are hence represented as time-varying. The usual method of implementing linear checksum

in this system fails due to the constantly varying system matrices. This necessitates the development of nonlinear error detection techniques with reduced redundancy.

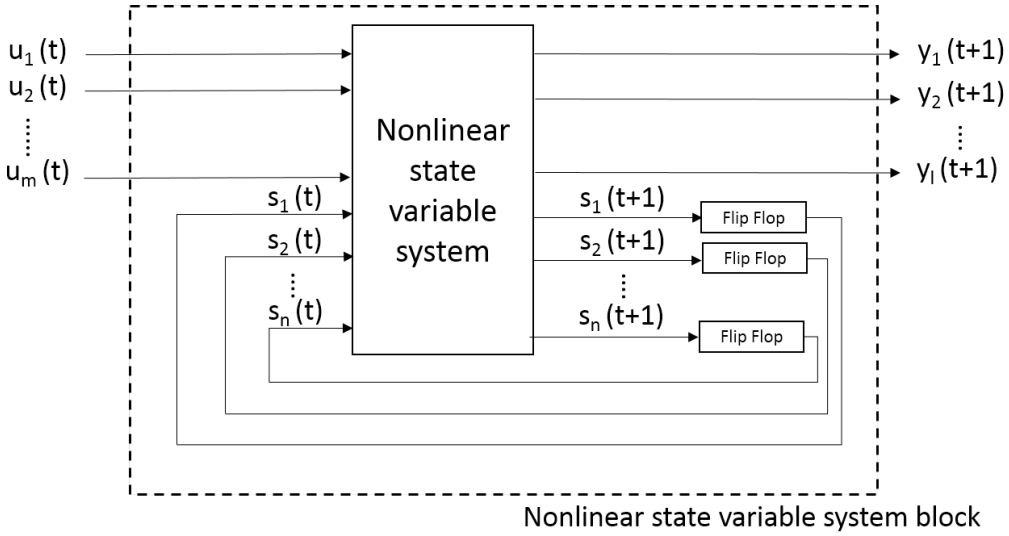


Figure 3.14: General structure of a nonlinear state variable system

3.2.3 Time-Freeze Linearization: A Brief Review

The brute force method to concurrent error detection in nonlinear digital circuits is to duplicate and compare the outputs between the duplicate circuit and the circuit under test. This approach is expensive and introduces very high hardware overhead. To reduce the cost of error checking, one possible approach is time-freeze linearization [71]. In time-freeze linearization method, the nonlinear digital circuit is modeled as a linear digital circuit in each time frame by *freezing* the values at circuit nodes corresponding to specified inputs of nonlinear circuit functions. The term *freezing* refers to treating the respective node values as constants over the width of a single time frame, even though these values change from one time frame to another. This facilitates the use of checksum codes, traditionally used in linear digital circuits for, concurrent error detection in non-linear digital circuits. The checking circuitry is economically designed, using tapped subfunctions from specific internal nodes of the non linear digital circuit. In a nonlinear system, the matrices $A(t)$ and $B(t)$ are constantly varying, thereby making the computation of matrices X and Y also

time-varying in the checksum circuitry. Time-freeze linearization allows one to design the checksum circuitry in a way such that at each time instant the matrices X and Y are computed by using tapped subfunctions of system states. Although, this approach is followed to reduce error detection cost by avoiding hardware and/or software redundancy, this complicates the design of the checksum circuitry. This is the prime motivation for a simpler solution that will be able to perform concurrent error detection in nonlinear digital systems at further reduced cost.

3.2.4 Linearized Checksum and Residue Prediction

The basic idea of the proposed approach is to separate the linear and nonlinear dynamics of the system. It has been mentioned in Chapter 2 that under fault-free conditions, the checksum error is ideally zero. However, if a checksum error signal for a nonlinear system is generated based on a least-squares linear estimate of the same system, it will be non-zero even under fault free conditions, since there will be estimation mismatch between linear and nonlinear system dynamics. This non-zero error, in fact, stems solely from the excitation of non-linearities in the circuit. A prediction mechanism predicts the next checksum value based on a history of past error samples. In presence of fault(s) in the system, the predictor cannot predict the irregularity experienced on the checksum error signal. Thus, the diagnosis is performed by continually comparing the predictor output and the actual linearized checksum circuit output and the residue between the two signals is the actual error signal.

Linearized Checksum Generation

The least squares linear estimate of a nonlinear system involves the generation of sufficient number of training pair of input-output samples. Let us consider the general nonlinear system in (3.12). We apply r different sets of input vectors $\mathbf{u_1}(t), \mathbf{u_2}(t), \cdots, \mathbf{u_r}(t)$ to the nonlinear system, one at a time. For each input vector $\mathbf{u_i}, \forall 1 \leq i \leq r$, the nonlinear system

is simulated for a fixed number of samples, say k samples starting from an initial system state. At each sampling time instant, the input to the system are m-dimensional input vector $\mathbf{u_i}(t)$ and n-dimensional previous system states $\mathbf{s}(t)$ to generate the n-dimensional new system states $\mathbf{s}(t+1)$. This can be perceived as a system with $(n+m)$-dimensional input vector $\mathbf{p}(t) = [\mathbf{s_{u_i}}(t)^\intercal \ \mathbf{u_i}(t)^\intercal]$ and n-dimensional output vector $\mathbf{q}(t) = \mathbf{s_{u_i}}(t+1)$ where the subscript $\mathbf{u_i}$ denotes the system state with input $\mathbf{u_i}$. Hence, for each input vector $\mathbf{u_i}$, k pairs of input-output samples $\mathbf{p}$ and $\mathbf{q}$ are generated. Across r different sets of input vectors, a total of $r \times k = b$ (say) pairs of input-output samples are generated.

The nonlinear state space equation in (3.12) can be differently expressed as

$$\mathbf{s}(t+1) = \mathbf{M}(t)\mathbf{z}(t) \tag{3.13}$$

where $\mathbf{M}(t) = [\mathbf{A} \ \mathbf{B}]$ and $\mathbf{z}(t) = [\mathbf{s}(t)^\intercal \ \mathbf{u}(t)^\intercal]$. Now, the linear least squares problem is posed as:

Given b sets of input-output pairs $\mathbf{p}_j$ and $\mathbf{q}_j$, $\forall 1 \leq j \leq b$, determine $\hat{\mathbf{M}}$, a linear estimate of $\mathbf{M}$, such that with the linear estimate $\hat{\mathbf{q}} = \hat{\mathbf{M}}\mathbf{p}$, the total error $\sum_{j=1}^{b} ||\hat{\mathbf{q}}_j - \mathbf{q}_j||_2$ is minimized. Once $\hat{\mathbf{M}}$ is determined, it can be decomposed to obtain $\hat{\mathbf{A}}$ and $\hat{\mathbf{B}}$, the linear estimates of the system matrix $\mathbf{A}$ and the input matrix $\mathbf{B}$. The algorithm to obtain $\hat{\mathbf{M}}$ is given in Algorithm 1.

Algorithm 1 Linear Least Squares

```
1: procedure LINEAR ESTIMATE
2:     H = [p_1, p_2, ..., p_b]^T
3:     Perform SVD of H = UΣV*
4:     Create Moore-Penrose Pseudo-Inverse G = VΣ^{-1}U*
5:     for do  j = 1 to n
6:         Create vector d = [q_1(j), q_2(j), ..., q_b(j)]^T
7:         Generate jth row of M̂ as G.d
```

Once $\hat{\mathbf{M}}$ is determined, $\hat{\mathbf{A}}$ and $\hat{\mathbf{B}}$ are separated out blockwise in the same manner as $\mathbf{M}$ is formed. The proposed approach is depicted in the block diagram of Figure 3.15. One of the primary advantages of the proposed approach is that the least squares computation is a

one-time design stage procedure to determine the *linearized checksum*. Hence, this saves considerable amount of computational overhead in comparison to time-freeze linearization where some of the nonlinear functions need to be replicated in hardware.

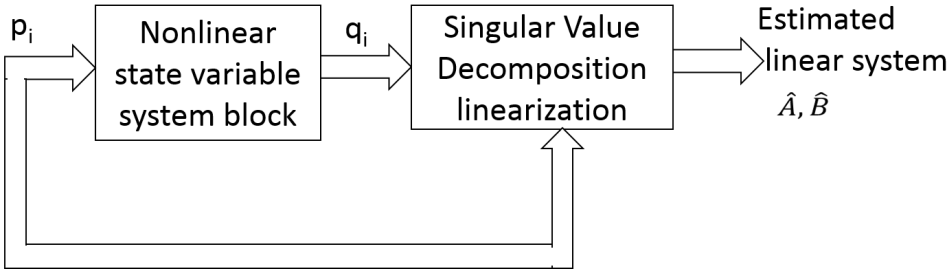


Figure 3.15: Block Diagram of the Linear Estimation Method

The selection of input vectors $\mathbf{u_i}, \forall 1 \leq i \leq r$ needs to be made carefully. For generation of training samples, the input vectors should be chosen from an input set which are expected to be applied to the nonlinear system during actual deployment. The least squares linear estimate of the system changes with the choice of input vectors. The reason for this behavior can be explained by the fact that different input vectors excite the nonlinearities of the system in different ways. A least squares estimate tries to fit a linear state-space trajectory of the system to the actual nonlinear trajectory that the circuit goes through. Depending on the choice of input vector, specific nonlinearities of the circuit can be activated and the nonlinear trajectory will be different. The choice of number of input vectors is guided by the tradeoff between nonlinearity coverage and linear overfitting. A very small set of input vectors will be insufficient to excite all nonlinearities and create an accurate linear estimate of the system while a large set of input vectors requires higher computational resources and may result in linear model overfitting. The same tradeoff dictates the choice of k - the number of samples generated from simulation with each input vector.

Once $\hat{\mathbf{A}}$ and $\hat{\mathbf{B}}$ are determined, the linear checksum module computes a linear checksum by selecting $\hat{\mathbf{X}} = \mathbf{CV}.\hat{\mathbf{A}}$ and $\hat{\mathbf{Y}} = \mathbf{CV}.\hat{\mathbf{B}}$. The check variable is computed as $\hat{c}(t+1) = \hat{\mathbf{X}}\mathbf{s}(t) + \hat{\mathbf{Y}}\mathbf{u}(t)$, thus generating the checksum error signal by subtracting this check signal from $\mathbf{CV}.\mathbf{s}(t+1)$. Unlike linear case, this difference will not be zero since

the state checksum computation is based on the linear estimate of the system. This nonzero error signal is fed into the linear predictive coding (LPC) unit for further processing.

Residue Prediction

Linear predictive coding (LPC) is a powerful predictive tool widely used in digital signal processing domain, mostly in speech and audio signal processing [111–114]. It denotes a mathematical operation where future values of a discrete-time signal are estimated as a linear function of previous samples. The most general representation is, given a discrete-time signal x with $(n-1)$ samples, the prediction for the nth sample is,

$$\hat{x}(n) = \sum_{i=1}^{p} a_i x(n-i) \tag{3.14}$$

where $\hat{x}(n)$ is the predicted signal, $x(n-i)$ the previous samples of the signal and a_i the coefficients of the predictor (FIR filter). The sample history used for the prediction is controlled by the parameter p. The objective is to minimize the estimate error $\epsilon(n) = \hat{x}(n) - x(n)$. This problem is well-studied in the field of linear algebra where the most common choice in optimization of parameters a_i is to solve the Yule-Walker autocorrelation (AR) equations derived by minimizing the expected value of the squared error $\mathrm{E}[\epsilon^2(n)]$. We use the well known Levinson-Durbin recursion method [115] to solve the Yule Walker equations.

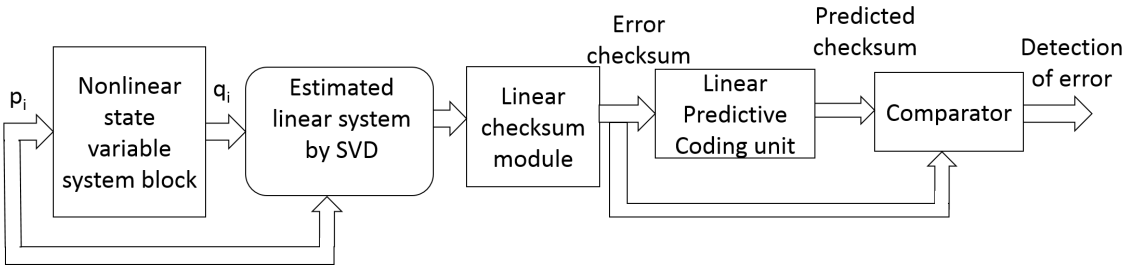


Figure 3.16: Block Diagram of Linearized Checksum and Residue Prediction Methodology

The linearized checksum error signal e is fed to the LPC unit as shown in Figure 3.16.

The LPC unit solves the Yule-Walker AR equations to obtain the FIR filter coefficients which are then used to predict the next checksum error signal $\hat{e}$. The actual checksum error signal in the next instant is compared with the predicted checksum error signal through a comparator and the difference between the two is referred to as predictor residue. This residue signal is used for error detection. In a fault-free system, the linearized checksum signal e is non-zero due to the mismatch between actual nonlinear system dynamics and linear estimation. Now, this non-zero error signal is predicted through LPC in successive time instants using past values of the checksum error. The difference between the predicted checksum and the actual checksum will again be non-zero due to prediction mismatch of the predictor. This prediction mismatch is controlled by the filter order of the LPC predictor and the sample history considered. However, in fault-free conditions, this prediction mismatch lies within a distinct threshold. In presence of fault(s) in the system, the actual checksum error shows a distinct discrepancy from the usual trend at the instant of fault occurrence and the predictor is unable to predict this unusual signal behavior from its past history. Therefore, the difference between two signals is higher than the chosen threshold in absolute magnitude and the fault is detected. Depending on the predictor parameters (filter taps and signal history length), the threshold changes. A higher filter order and longer signal history will lower the threshold by reducing prediction mismatch and improve performance at the cost of resource overhead.

It can be argued that linear predictive codes can directly be applied to the sum of all the states of a digital system to remove the computational cost of system linearization. Under error-free conditions, the sum of system states follows a definite trajectory which can similarly be predicted by LPC. In presence of errors, the sum of the states deviate from the usual trend and the predictor residue will detect the error. However, this scheme will suffer from practical drawbacks. The sum of system states results in very large dynamic range of signals and hence the checking circuit needs to have higher dynamic range adding to the hardware resource cost. On the other hand, the predictor residue in our proposed

approach has a small value which can be resolved using circuits with lower precision.

3.2.5 Simulation Results

We perform simulation experiments on a nonlinear Volterra filter consisting of 13 multipliers and 10 adders whose state-space representation is given by equation (4.9) [71], in MATLAB environment.

$$\begin{bmatrix} s_1(t+1) \\ s_2(t+1) \\ s_3(t+1) \\ s_4(t+1) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0.183 & -0.946 & F_{02}+0.197 & F_{12}-0.241 & 0.1 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} s_1(t) \\ s_2(t) \\ s_3(t) \\ s_4(t) \\ u(t) \end{bmatrix} \quad (3.15)$$

where $F_{02}(t) = -0.971s_1(t) + 0.946s_2(t) + 0.552u(t)$ and $F_{12}(t) = 0.183s_1(t) + 0.072s_2(t) + 0.298u(t)$. Thus the system matrix of this filter contains polynomial functions of the states and input signal and continuously varies over time.

One of the primary uses of a Volterra filter is in the field of digital telecommunication where it is used as a predistorter to compensate for intermodulation distortion in devices like power amplifiers and frequency mixers. Hence, in our experiments we applied orthogonal frequency-division multiplexing (OFDM) signals as input $\mathbf{u}(t)$ to the system. We chose 64-QAM modulation scheme with a carrier frequency of 1 GHz to generate an OFDM input signal. We chose a total of $r = 4$ different digital bit sequences to generate different OFDM stimuli which are then used as inputs to the Volterra filter of (4.9) to generate the data pair $\mathbf{p}$ and $\mathbf{q}$. The simulation with each input is performed over 1000 time samples. Using these data, we generate the least-squares linear fit to the system in (4.9) by the SVD linearization

method mentioned in Section 3.2.4. The linear estimate of the system is obtained as,

$$\hat{\mathbf{M}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0.1824 & -0.9158 & 0.2062 & -0.3389 & 0.1086 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{3.16}$$

We use this matrix to compute the linearized checksum signal of SCC. The nominal coding vector chosen in this simulation is $\mathbf{CV} = [0.25\ 0.25\ 0.25\ 0.25]$. It is useful to select CV in a way such that the constraint $\sum_{i=1}^{n} |\mathbf{CV}(i)| \leq 1$ is satisfied to avoid overflow errors[68]. However, it should be mentioned that fault coverage changes with the choice of coding vector which we demonstrate later. The optimal choice of coding vector is a topic of future research and is beyond the scope of this paper.

For fault injection, we study two kinds of faults: those that cause single data bits of state output data word to be different from their correct values and those that cause multiple data bits to be incorrect. The first is commonly referred as a *bit error* and the latter is called a *word error*. We discretize each output state with a word size of W bits with 1 bit reserved for signed number representation. Among the $(W-1)$ bits, $\frac{W}{2}$ bits are used to represent the decimal portion of the number. Bit errors are injected into the system by selecting a state at random (with equal probability) and then by complementing any one bit chosen at random where each bit has equal probability of being complemented. Similarly, word errors are injected by either complementing or not complementing each bit of the data word randomly selected from the n states, the probability of bit complementation being 0.5.

A nominal fault-free simulation is performed using the linearized checksum. The predictor selected for our simulation has a filter order of 4 and uses a past history of 8 samples. As mentioned in Section 3.2.4, an error is detected only if the difference between LPC predictor output and the linearized checksum signal is above a fixed threshold. This threshold is due to three main reasons: 1) Estimate mismatch between actual nonlinear dynamics and

linearized checksum, 2) Prediction mismatch between LPC predictor and the checksum error signal, and 3) Quantization noise due to finite precision arithmetic. From nominal simulations, the linearized checksum signal is seen to be less than 0.05 in magnitude without any injected errors and hence this threshold level of 0.05 was chosen to be the error detection threshold during actual error injection. If a different nonlinear system is considered, this threshold will be different and needs to be determined from nominal fault-free simulations.

In a linear time-invariant system, errors can be injected every alternate time frame, since the errors do not accumulate over time as far as linear check variables are concerned. Though finite precision arithmetic errors and injected soft errors accumulate between times t and $t+1$, both the state checksum and the data checksum circuitry operate on the same data at next instant $t+1$. However, in a nonlinear time-varying system, the effects of an injected soft error take a few clock cycles to decay and determined by the effective time-constant of the excited non-linearities. We determined this time-constant to be 15 cycles in our simulation setup under the mentioned conditions. Hence, both bit errors and word errors are injected every 15 clock cycles. The number of least significant bits affected during an error injection is referred to as the *injection range*. We conducted experiments in which bit errors are injected into all W bits of a data word, and then into only the least significant $\frac{W}{2}$, $\frac{W}{4}$ and $\frac{W}{8}$ bits, respectively. The same experiments are also run using time-freeze linearization method. The results of our experiment with different word sizes for both bit errors and word errors are shown in Figures 3.17 and 3.18.

In figures 3.17 and 3.18, a lower value of ratio between injection range and data word length indicates error injection in LSBs whereas a higher value indicates error injection in bits of higher significance. From the results, it is clearly seen that bit errors in the least significant bits are harder to detect in both the proposed approach and time-freeze linearization. However, we demonstrate that our proposed approach is superior in detection of bit errors whereas it performs on a similar scale for word errors. Since word errors, in

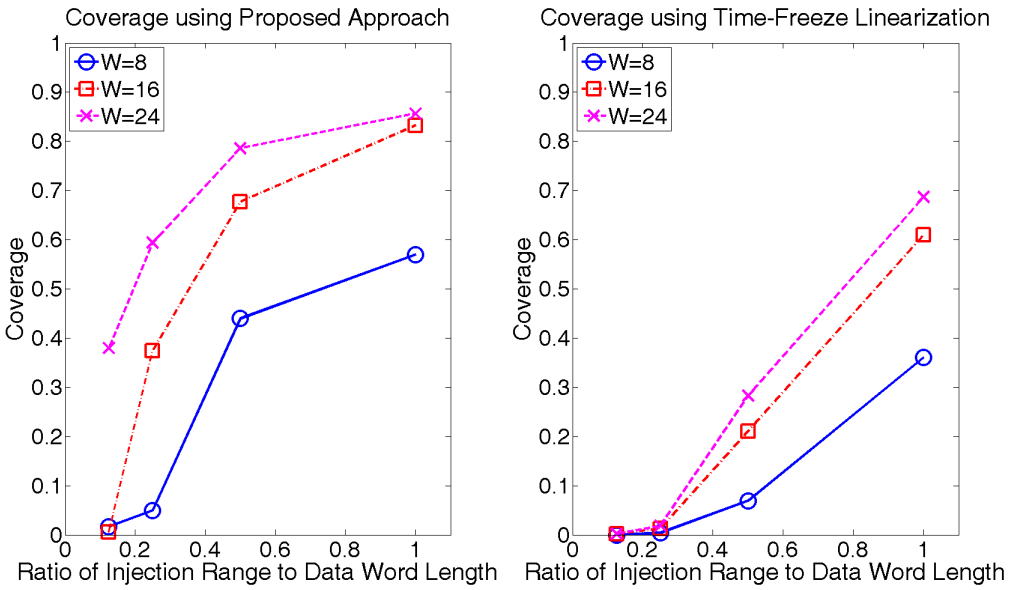


Figure 3.17: Fault Coverage vs Injection Range for Bit Errors

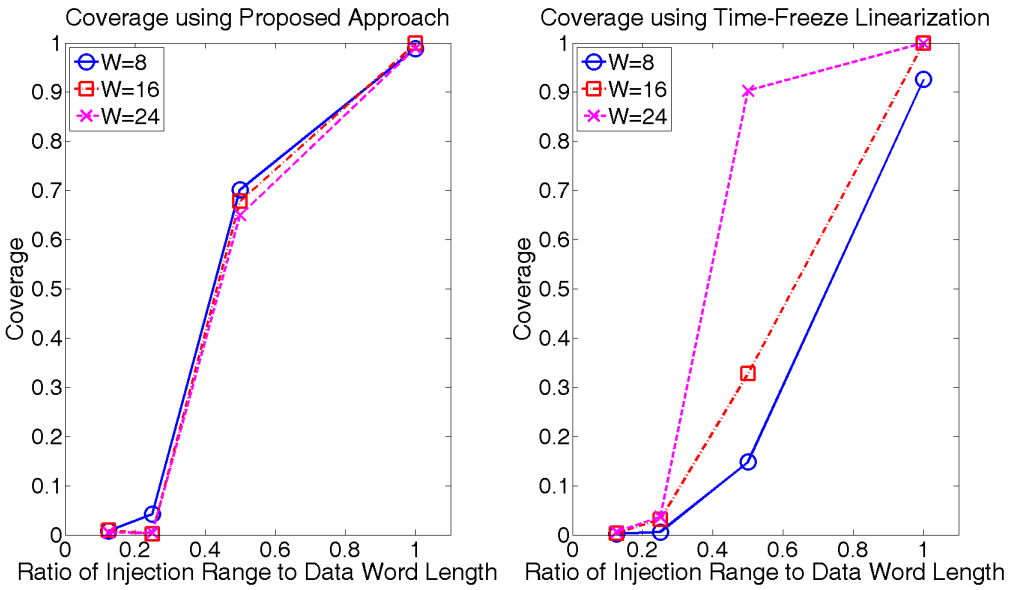


Figure 3.18: Fault Coverage vs Injection Range for Word Errors

general, have multiple bit errors, these are easier to detect than bit errors. The coverage increases as the word length is increased since it allows better resolution for faults to get detected. The predictor residue signal in the nominal system and with single error injection are shown in Figure 3.19. The low value of the signal allows us to use checking circuit with reduced precision and we use a precision of W/2 bits for a word length of W bits.

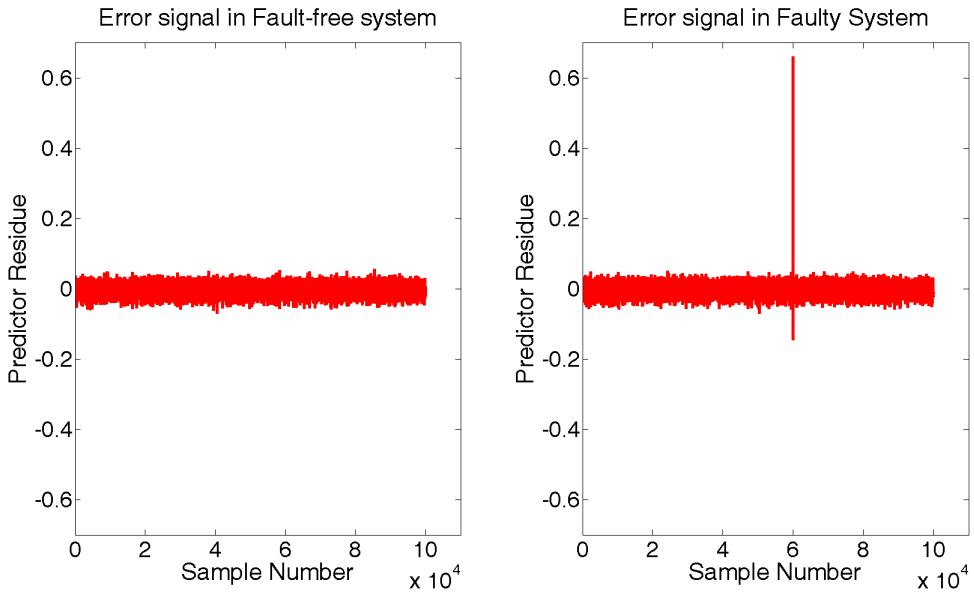


Figure 3.19: Comparison of Predictor Residue without and with Error Injection

Table 3.1 indicates the variation of error coverage with different choices of coding vectors with a word length of 8 bits. The 2 coding vectors CV_1 and CV_2 are given as, CV_1 = [0.25 0.25 0.25 0.25] and CV_2 = [0.15 0.15 -0.4 0.25]. The results clearly show that future research is needed to optimize the choice of coding vector. Different sets of coding vectors control the sensitivity of the contribution of system states to the checksum error line.

Table 3.1: Error Coverage with different Coding Vectors

Injection Range	Bit Error Coverage		Word Error Coverage	
	CV_1	CV_2	CV_1	CV_2
1 bit	0.0166	0.0182	0.0082	0.0155
2 bits	0.05	0.0524	0.0421	0.0791
4 bits	0.44	0.32	0.7015	0.68
8 bits	0.57	0.62	0.8566	0.79

3.2.6 Summary

In this chapter, a new methodology is described for detection of soft errors in nonlinear digital state variable systems. The proposed technique generates a least-squares linear fit to the nonlinear system with the aid of singular value decomposition (SVD). Linearized

checksum codes are derived from the linear system estimate which are then used to predict the error signal using linear predictive codes. It is demonstrated that the predictor residue signal can be used for fine-grained error detection through a simulation of a nonlinear digital Volterra filter. It is further shown that this methodology outperforms time-freeze linearization in terms of error detection capability at reduced overhead, since the latter needs duplication of the system in the worst case. The experimental analysis for both single bit errors and multiple bit errors illustrate the efficacy of the proposed approach. The presented results indicate the effectiveness of the proposed technique in dealing with soft errors in nonlinear digital state variable filters.

CHAPTER 4

CONCURRENT ERROR DETECTION IN NONLINEAR CONTROL SYSTEMS

This chapter proposes a machine learning assisted methodology for error detection in non-linear control systems operating under arbitrary failure mechanisms.

4.1 Proposed Error Detection Methodology

4.1.1 Nonlinear Control System

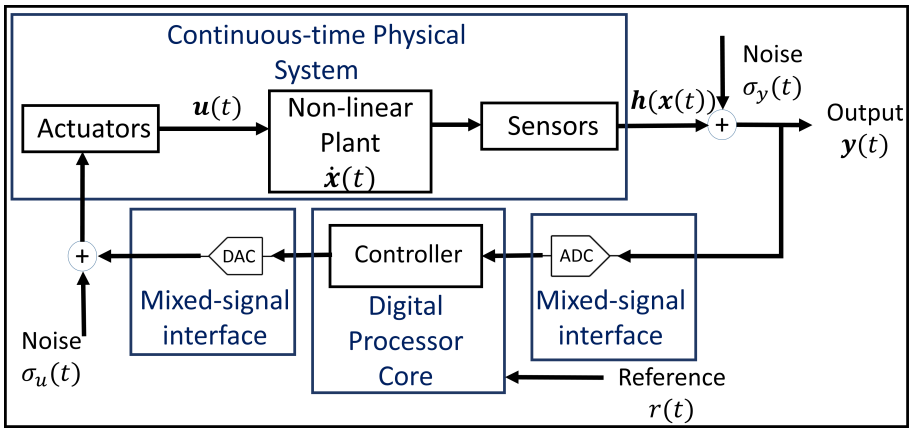


Figure 4.1: Representation of a nonlinear state space control system

A nonlinear system architecture is illustrated in the block diagram of Figure 4.1. The nonlinear plant encapsulates the dynamical equations of the system behavior. The sensors measure the outputs of the plant that are certain combinations of the system states. The controller is implemented on a digital processor core along with appropriate mixed-signal interfaces. The controller performs its algorithmic computations based on the digitized sensor measurements and external reference input. The digital control signal is applied to the actuators through DACs, that impart the necessary action to the plant. Uncorrelated noise sources are considered in sensor readings and actuation signals.

The nonlinear dynamics of the plant is expressed by the differential equation:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)) \tag{4.1}$$

where $\mathbf{x}(t) \in \Re^n$ and $\mathbf{u}(t) \in \Re^m$ represent the $n \times 1$ state vector and $m \times 1$ control vector respectively. The vector valued function $\mathbf{f}(\mathbf{x}, \mathbf{u}) = [f_1(\mathbf{x}, \mathbf{u}), f_2(\mathbf{x}, \mathbf{u}), \cdots, f_n(\mathbf{x}, \mathbf{u})]$ denotes the nonlinear relationship between plant states and control inputs. The measurement equation of the plant is expressed as:

$$\mathbf{y}(t) = \mathbf{h}(\mathbf{x}(t)) + \sigma_{\mathbf{y}}(t) \tag{4.2}$$

where $\mathbf{y}(t) \in \Re^p$ represents the $p \times 1$ output vector of the system. The function $\mathbf{h}(\mathbf{x}) = [h_1(\mathbf{x}), h_2(\mathbf{x}), \cdots, h_p(\mathbf{x})]$ denotes a general nonlinear combination of the system states that form the measurements. The measurement noise is represented by the $p \times 1$ vector $\sigma_{\mathbf{y}}(t)$.

The controller computes the appropriate control signal based on the reference input $\mathbf{r}(t)$ and observed measurements $\mathbf{y}(t)$ as:

$$\mathbf{u}(t) = \mathbf{g}(\mathbf{y}(t), \mathbf{r}(t)) + \sigma_{\mathbf{u}}(t) \tag{4.3}$$

where $\mathbf{g}(.)$ implements the nonlinear control law and $\sigma_{\mathbf{u}}(t)$ is the actuator noise. Typically, the control laws require state values that are estimated from the observed measurements using state estimation techniques such as extended Kalman filter. Here it is assumed that the controller implementation $\mathbf{g}(.)$ incorporates such state estimation techniques.

4.1.2 State Encoding Based Detection Methodology

The proposed approach implements a sliding temporal window over past system measurements and control inputs to predict an encoded check state that is compared with information obtained from present system data to detect any anomalies. In our methodology, an additional state known as *mapped predictive check state* (MPCS) is computed at any point of time t from known system data at times $t, t-1, t-2, \cdots$ using a nonlinear prediction function $F_c(.)$. At time $t+1$, a different nonlinear function $F_m(.)$ is used to predict the MPCS from the latest system data. The state computed using $F_m(.)$ is called *mapped information state* (MIS). The functions $F_c(.)$ and $F_m(.)$ are optimized in such a way that there is minimal difference between the MPCS and MIS in fault-free nominal operation while any abnormal behavior causes this difference to exceed a pre-defined threshold.

Mathematically, let the reference $\mathbf{r}(t)$, measurement $\mathbf{y}(t)$ and the control input $\mathbf{u}(t)$ are unified in a single variable $\mathbf{s}(t)$ as

$$\mathbf{s}(t) = \begin{bmatrix} \mathbf{r}(t) \\ \mathbf{y}(t) \\ \mathbf{u}(t) \end{bmatrix} \tag{4.4}$$

Similarly, the measurement $\mathbf{y}(t)$ and the control input $\mathbf{u}(t)$ are unified in a single variable $\mathbf{z}(t)$ as

$$\mathbf{z}(t) = \begin{bmatrix} \mathbf{y}(t) \\ \mathbf{u}(t) \end{bmatrix} \tag{4.5}$$

After observing the system data at time t, the MPCS for instant $t+1$ is formed by the nonlinear mapping $F_c(.)$ of data over T_p time samples, that define the predictive horizon,

as:

$$s_c(t+1) = F_c(\mathbf{s}(t), \mathbf{s}(t-1), \cdots, \mathbf{s}(t-T_p+1)) \tag{4.6}$$

Similarly the MIS is generated at instant $t+1$ from the system data $\mathbf{z}(t+1)$ by the other nonlinear mapping as:

$$s_m(t+1) = F_m(\mathbf{z}(t+1)) \tag{4.7}$$

The error signal $e(t+1)$ is computed as the difference between the MPCS $s_c(t+1)$ and the MIS $s_m(t+1)$ as:

$$e(t+1) = s_c(t+1) - s_m(t+1) \tag{4.8}$$

Equations (4.6), (4.7) and (4.8) demonstrate that the error signal depends on the prediction functions $F_c(.)$ and $F_m(.)$ and the predictive time horizon T_p that indicate the number of data samples considered while computing the MPCS $s_c(t)$. The mapping functions $F_c(.)$ and $F_m(.)$ are implemented in a way such that $e(t)$ lies below a certain threshold for fault-free systems. The block diagram of the proposed scheme is shown in Figure 4.2. Next, we study the selection of the optimal value of T_p and investigate the choice of prediction functions $F_c(.)$ and $F_m(.)$.

4.1.3 Predictive Time Horizon Selection

The discussion about predictive time horizon stems from the Markov properties of systems. In a Markov Decision Process (MDP), only the current state affects the next state, or

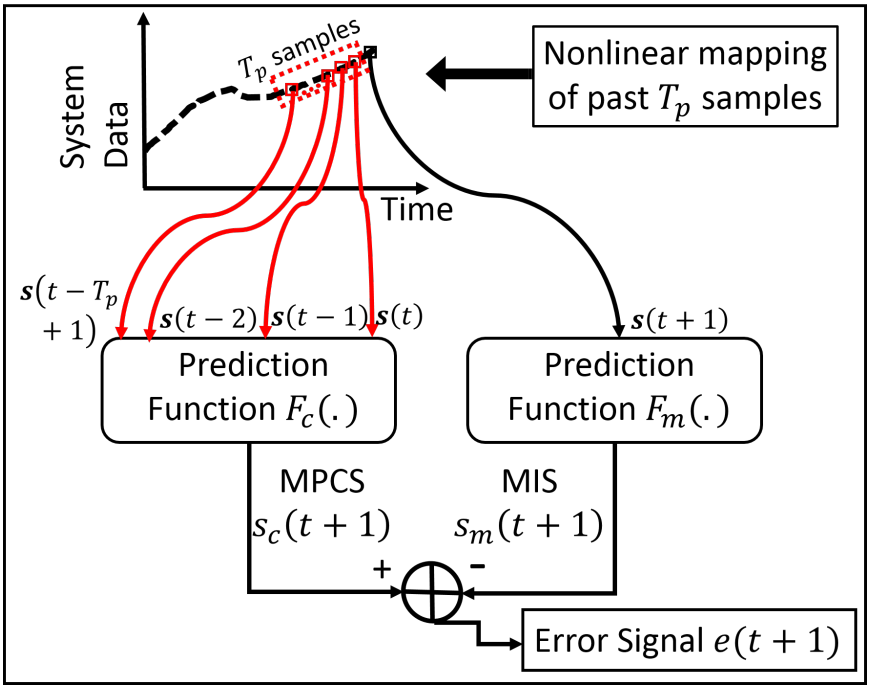


Figure 4.2: Real-time error detection methodology using nonlinear mapping functions over a time horizon of T_p samples - the error signal $e(t+1)$ is formed as the difference between the MPCS $s_c(t+1)$ and MIS $s_m(t+1)$

in other words, the future is conditionally independent of the past given the present state. This means that the evolution of $\mathbf{x}(t)$ is based solely on the previous state $\mathbf{x}(t-1)$ and the action taken $\mathbf{u}(t-1)$. Although this assumption seems theoretically valid from (4.1), it is somewhat unrealistic, as the accurate computation of $\mathbf{u}(t-1)$ requires the states to be *fully observable* that is seldom true in practical systems. A generalization of MDPs are partially observable MDPs (POMDPs), in which the present observation $\mathbf{y}(t)$ is dependent on the current state $\mathbf{x}(t)$ and the previous action applied. Additional noise sources in the measurement process further invalidate the perfect MDP assumption in control systems. POMDPs are typically tackled using information integration over time, where the true dynamics of the system is not directly revealed from a single observation, but gradually form over multiple observations at different time instants. Since, full system observability is not assumed, the temporal mapping needs past samples to represent the memory effect and actual nonlinearity in the system. A low value of T_p does not capture enough information

about the state trajectory to accurately represent the system dynamics. In addition, it is not robust towards noise as reliance on a few data samples results in the assimilation of corrupted information. However, a low value of T_p reduces the computational overhead while computing the check state $s_c(t)$. On the contrary, a high value of T_p has high robustness towards noise and external disturbances at the cost of increased computational burden. This requires a careful analysis of the optimum choice of T_p.

The primary objective of mapping over a temporal window is to learn the system dynamics from consecutive observations. This is tantamount to solving a nonlinear system of equations for the unknown system model parameters and the number of observations required is equal to the number of variables to be determined. It has been mentioned in Section 4.1.2 that a separate function $F_c(.)$ is used to learn the nonlinear dynamics. Hence, it is important to choose the maximum degree of equivalent polynomial nonlinearity that can best represent the system dynamics while training the mapping function $F_c(.)$. Let the maximum degree of equivalent polynomial nonlinearity that needs to be captured is d. It is known from mathematical approximation theory [116] that for well-behaved functions, there exists an dth degree optimal polynomial that can interpolate $(d+1)$ points with the corresponding error curve oscillating between $+\epsilon$ and $-\epsilon$ for a small finite $\epsilon \in \Re$. In addition, due to partial state observability from n states to p measurements where $(p \leq n)$, there is a subspace projection from $\Re^n \rightarrow \Re^p$, thus increasing the number of parameters to be estimated. We define this with the *state reduction factor* $f_{n \rightarrow p}$ as $f_{n \rightarrow p} = \dfrac{n}{p}$. Keeping this discussion in mind we make the following proposition:

Proposition I: To map out a maximum degree of nonlinearity equivalent to a dth degree polynomial, the least number of observations required with a state reduction factor of $f_{n \rightarrow p}$ is $T_{p(min)}) = f_{n \rightarrow p} \times (d+1)$.

This proposition excludes the effects of noise on the observed measurements. In presence of sensor and actuator noise, the observations are corrupted and the extent of unreliability depends on the noise variance. Hence, for practical purposes the actual value of

T_p is increased considering the noise corruption factor. Thus, the final choice of optimal temporal window length is $T_p = T_{p(min)} \times f_{noise}$ where $f_{noise} \geq 1$ takes into account the uncertainty present in observations due to noise sources and selected based on the knowledge of noise statistic of the system.

4.1.4 Mapping Function Choices

Possible choices of prediction functions explored in this work are divided into 3 categories: i) Volterra series, ii) Machine Learning algorithms and iii) Analytical functions.

i) Volterra series: In [117], Boyd and Chua demonstrated that any general nonlinear system behavior exhibiting fading memory can be approximated to arbitrary accuracy using a truncated Volterra series. A truncated Volterra series of order N is expressed as:

$$y[n] = \sum_{r=1}^{N} \sum_{k_1=0}^{M-1} \cdots \sum_{k_r=0}^{M-1} h_r[k_1, \cdots, k_r] x[n-k_1] \cdots x[n-k_r] \tag{4.9}$$

where $h_r[k_1, k_2, \cdots, k_r]$ is the rth order Volterra kernel, N represents the maximum order of the Volterra series, M denotes the memory for each of the kernel, $x[n]$ and $y[n]$ are the input and output of the truncated Volterra series. It is well known [118] that kernel estimation is an ill-posed problem and the number of parameters representing the kernels increases geometrically with the kernel order. This leads to truncation of Volterra series to $N = 4$ where higher terms are rarely used in practice. In our case, the memory of each kernel M is synonymous to the temporal observation window T_p. In the pre-deployment design stage, the Volterra series is presented with a shifting data set of T_p past observations as $x[n]$s and another shifting data set of present observations as $y[n]$s for kernel estimation. Once the Volterra kernel is learned using the design stage training data set, it is deployed as the mapping function $F_c(.)$ for error detection.

ii) Machine Learning algorithms: Two possible machine learning algorithms are appro-

priate for this purpose - a) regression functions and b) neural networks. In this work, we explore Multivariate Adaptive Regression Splines (MARS) as the regression function. This is a nonparametric regression technique that builds models of the form:

$$\hat{f}(x) = \sum_{i=1}^{k} c_i B_i(x) \tag{4.10}$$

where c_is are model coefficients and $B_i(x)$s are basis functions. The MARS model building process performs the learning of the coefficients and the basis functions when presented with a training data set formed in a similar fashion as in the Volterra series training.

For neural network implementation, we choose Adaptive Resonance Theory (ART)- a neural network model developed on the aspect of how the brain processes information. Based on past observed data, an ART model is built up whose output known as 'expectation' is compared with the present observation known as 'sensation'. The difference between sensation and expectation is minimized such that it does not exceed a set threshold called the 'vigilance' parameter. Typically the basic ART algorithm is an unsupervised learning model. In this work, we use ARTMAP [119] that combines two slightly modified ART-1 (accepting only binary inputs) and ART-2 (accepting continuous inputs) units into a supervised learning structure. Using ART as possible neural network model provides an additional benefit of continuous post-deployment learning of the prediction functions since all possible scenarios encountered by a cyber-physical system cannot be envisaged in pre-deployment design simulation stage.

iii) Analytical functions: Finally, analytical expressions can be derived from the system models (4.1), (4.2) and (4.3) to compute the MPCS. Though this approach appears to be computationally favorable since it doesn't require any training phase, calculating such an analytical function for representing the system dynamics is often an infeasible task for highly complex systems and has not been explored in this work.

4.1.5 Implementation

In the pre-deployment design and analysis phase, the learning/training of the prediction functions is achieved. However, simulation of all possible input conditions and estimating all foreseeable environmental scenarios are implausible in real life. Hence, in the initial phase of post-deployment period further learning and calibration of these prediction functions can be conducted before real-time error detection can be performed more accurately. The proposed methodology is implemented in software, typically on an embedded digital processor that also contains the control algorithm for a control system. Thus, the power consumption of the proposed scheme is determined by the clock cycles and memory overhead requirement that are reported in the results.

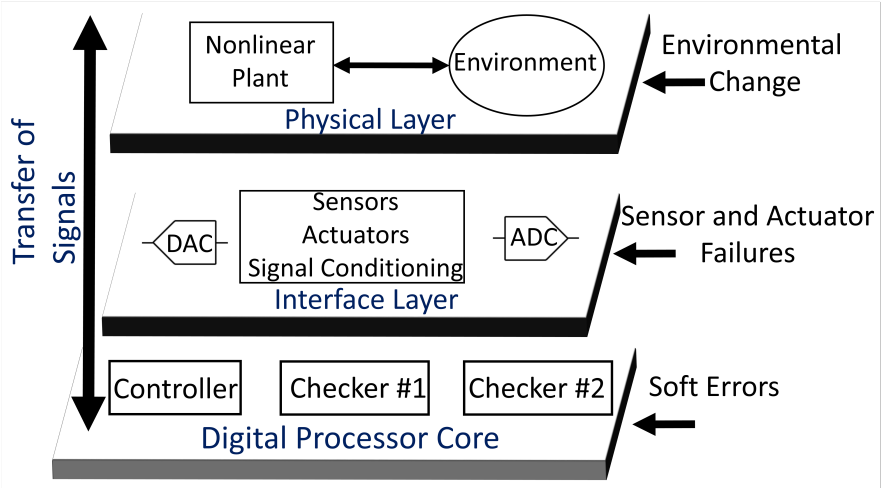


Figure 4.3: Implementation of proposed checking methodology in a cross-layer hierarchy along with scope of considered faults

The implementation of the proposed checking mechanism across different abstraction levels is illustrated in Figure 4.3. The physical layer consists of the actual nonlinear continuous time system interacting with the real environment. The mixed-signal interface layer contains the sensors and actuators interacting with the physical system along with the appropriate transducers, converters and signal conditioning circuit. The signals from this layer interface with the underlying embedded digital processor core where the control

algorithms are typically implemented. The proposed checking methodology also runs as an additional software program on the digital core. The fault models considered in this work, discussed in detail in Section 4.2, expose the mixed-signal layer and the digital core to errors. The sensors and actuators are amenable to physical malfunctions or extraneous signal coupling resulting in compromise of their nominal operation while the processor core is subject to soft errors in the form of transient bit-flips due to particle strikes. Since the proposed checker also runs on the same core, the detection is vulnerable to these soft errors and its accuracy can deteriorate due to computation mistakes. Implementation of one checker instance can result in false positives (triggering of error detection without any actual controller fault). Hence, we implement two checker instances and compare the error signals of these two checkers to generate a final error signal. The probability of bit flips affecting the two checkers identically and masking their individual detection is low, thus rendering the implementation robust.

4.2 Fault Models

In this work, we study cross-layer failure mechanisms and deal with errors/anomalies in all three layers - physical, interface and digital and investigate the detection capabilities of the proposed methodology.

4.2.1 Physical Layer

The state equation in (4.1) describes how the system behaves when operating in a particular environment. With change of operating environment, the system dynamics is altered that is modeled by the change in the function $\mathbf{f}(\mathbf{x}, \mathbf{u})$ in (4.1) as:

$$\dot{\mathbf{x}}(t) = \mathbf{f}'(\mathbf{x}(t), \mathbf{u}(t)) \tag{4.11}$$

where $\mathbf{f}'(.)$ is the modified system dynamics due to a change in operating environment. Instead of arbitrarily changing the system model parameters from $\mathbf{f}(.)$ to $\mathbf{f}'(.)$, the environment in which the plant operates is changed by appropriately modifying the state evolution model. For example, the modeling of a braking event of a vehicle on a level slope is different than the braking on a downward/upward slope. We consider training the prediction function on measurements from one system model and test the detection methodology on a different system model mimicking a shift in the operating environment. In such cases, instead of error detection the checking methodology performs detection of events that the system is not trained to handle - henceforth referred to as '*anomaly*'. The choice of neural network topology as the prediction function is particularly helpful since it can undergo further on-field learning to incorporate the new data and learn about unmodeled situations for improved detection capabilities.

4.2.2 Interface Layer

Among the blocks in the interface layer that interact with the physical plant, we consider failures in sensors and actuators. The sensor/actuator malfunctions are modeled as i) additive signals and ii) parametric perturbations. Any control action computed based on untrustworthy sensor data is erroneous and negatively impacts the plant performance. Similarly, a malfunctioning actuator degrades the plant behavior by imparting incorrect control force. The sensor/actuator fault model is illustrated in Figure 4.4.

i) Additive signals: These are extraneous signals that alter the output of a sensor/actuator. This is modeled by an additive signal $\Delta\mathbf{z}(t)$ to the output $\mathbf{z}(t)$ of a sensor/actuator that corrupts the sensed data or alters the applied control action. In this work, such an effect is represented either as a step signal for modeling of permanent DC offsets or persistent signal coupling. These additive signals are different from the noise signals $\sigma_{\mathbf{u}}(t)$ and $\sigma_{\mathbf{y}}(t)$ with known statistics.

ii) Parametric perturbations: Field degradation of sensors and actuators are modeled as

parametric perturbations from the nominal r-dimensional parameter set $\{w_1, w_2, \cdots, w_r\}$ to the modified set $\{\tilde{w}_1, \tilde{w}_2, \cdots, \tilde{w}_r\}$. This parametric modification alters the transfer function of sensors/actuators in a way that is not well modeled by additive signals alone.

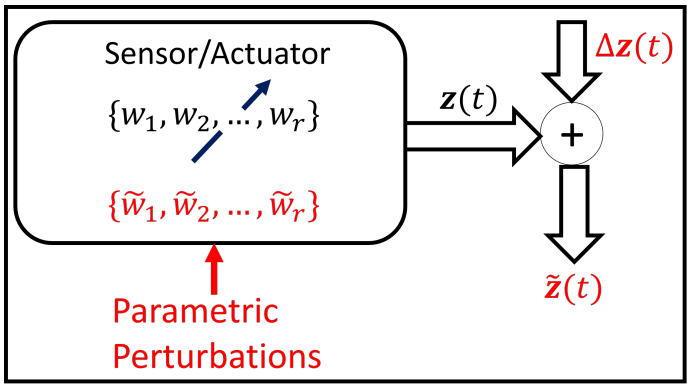


Figure 4.4: Fault model of sensor and actuators include parametric perturbations and injection of corrupting signals

4.2.3 Digital Layer

The fault model in the digital processor core shown in Figure 4.3 involves injection of soft errors in the form of bit-flips in the different blocks of the processor on which the control algorithm is implemented. In this work, we study soft errors in both software and hardware implementation.

i) Software Error Model: Fault injection experiments on flip-flops (FFs) in the RTL implementation of a processor core provide the best evaluation of effects of bit-flips on high level program execution. However, in comparison with lower level approaches, fault injection on microarchitectural structures modeled in performance simulators, is orders of magnitude faster and proves to be an effective method for fault detection analysis. Among the publicly available full system simulators, we chose Gem5 [120] as the system simulator for two primary reasons - i) it is *cycle-accurate*, thus providing per cycle granularity of fault injection, ii) it includes all *key microarchitectural components* that model the hardware arrays on which faults of any duration and severity can be injected. The simulator

configuration that is chosen is shown in Table 4.1.

Table 4.1: Simulator Configuration

Configuration Parameters	**Values**
ISA Configuration	x86
Pipeline	OoO (Out-of-Order)
Physical Register File	256 Integer 128 Floating-point (FP)
Issue Queue Entries	32
Load/Store Queue Entries	16 Load 16 Store
Reorder Buffer Entries	40
Functional Units	2 Integer ALUs 2 FP ALUs
L1 Instruction Cache	32 KB, 64-B line 128 sets, 4-ways write back policy
L1 Data Cache	32 KB, 64-B line 128 sets, 4-ways write back policy
L2 Cache	1 MB, 64-B line 1024 sets, 16-ways write back policy
Branch Predictor	Tournament predictor
Branch Target Buffer	Direct-mapped 2K entries

The soft error fault model is classified into two types - i) **transient** where a storage element's bit value is flipped in one clock cycle of the program execution and ii) **permanent** where a storage element's bit value is permanently fixed at either '0' or '1' representing a stuck-at fault. These bit flips are further subdivided into *bit error* - single bit-flips and *word error* - simultaneous multiple bit flips. The fault effects that are explored in this work are categorized into 4 classes:

1. **Masked** - The fault injection does not affect the program execution and the fault-

injected result is identical to the nominal fault-free simulation data.

2. **Silent Data Corruption (SDC)** - In this fault injection, the program execution completes normally but the output data is corrupted and differ from fault-free results.

3. **Timeout** - This includes situations where the program flow is trapped due to fault injection and is unable to commit any further instructions and the program terminates after a pre-configured execution timeout limit.

4. **Crash** - This represents situations where the fault insertion results in an unrecoverable situation and hence the simulated program crashes and terminates abruptly.

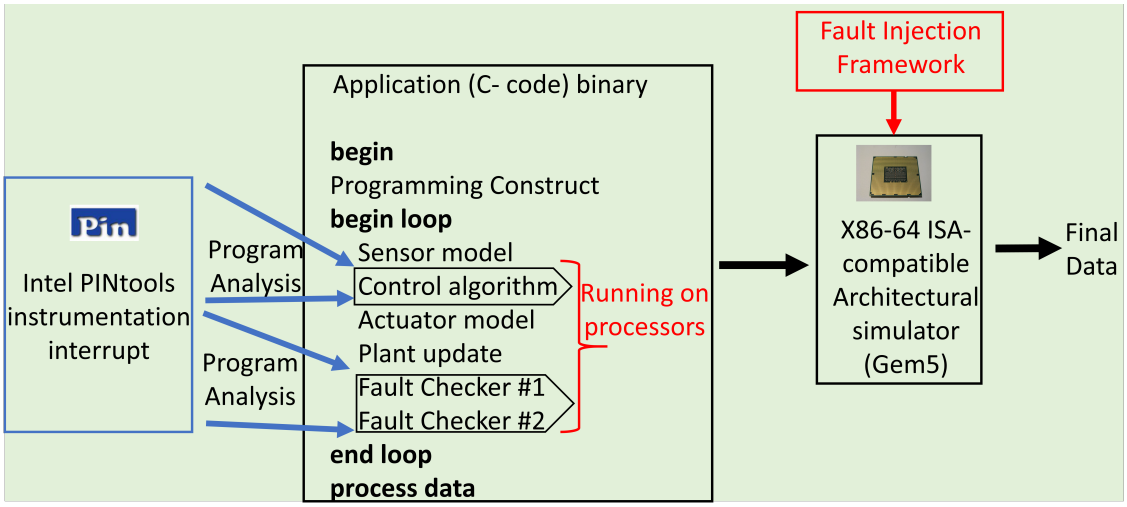


Figure 4.5: Fault injection infrastructure for simulation of soft errors in computation

As illustrated in Figure 4.5, the application code contains an iterative programming construct that represents receiving sensor data, applying the control algorithm to compute optimal control signals and imparting the desired action to the plant through the actuator model at each iteration. The plant equations are invoked to simulate plant behavior and fault-checking is performed by two checkers. As shown in Figure 4.3, only the control algorithm and the fault detection methodologies are executed on the digital processor since the rest of the algorithm mimics the actual physical behavior of the system. Hence,

caution is exercised while injecting errors such that it only affects the execution of the control algorithm and checkers while keeping execution of rest of the program (representing plant operation) fault-free. For this purpose, we use a dynamic binary instrumentation tool called Pin [121] developed and supported by Intel and supplied free for non-commercial use. Using this tool, the trace analysis of an executable program is performed so that exact instructions pertaining to the control and the checker algorithms can be identified in the entire instruction set of the program trace. This tool is further used for isolating the computation overhead of the proposed checking mechanism from the entire program's resource requirement by inserting Pintool instrumentation routines. The fault injection mechanism is adopted from [122] where a *fault mask* is used for fault insertion. A fault mask specifies the following:

1. the processor core in which the fault is to be injected
2. the microarchitectural structure on which the fault is injected
3. bit position(s) of fault injection
4. exact instruction(s) of fault injection
5. the fault type that determines whether it is a transient or a permanent fault

The microarchitectural structures that are injected with faults are load/store queue, issue queue, int and FP register file and tag and data portion of both L1 and L2 caches. The compiled application is executed on the x86 core through the Gem5 simulator along with the appropriate fault mask configuration and the results are analyzed to identify detection performance of the proposed checking methodology.

ii) Hardware Error Model: A hardware model for a soft error is one that implements a bit flip in the Register Transfer Level (RTL) or gate level design. In this work we use a soft error injection infrastructure for ARM AMBER processor [123]. The AMBER core is a 5-stage ARM pipeline synthesized to an FPGA and runs at 40 MHz frequency without any

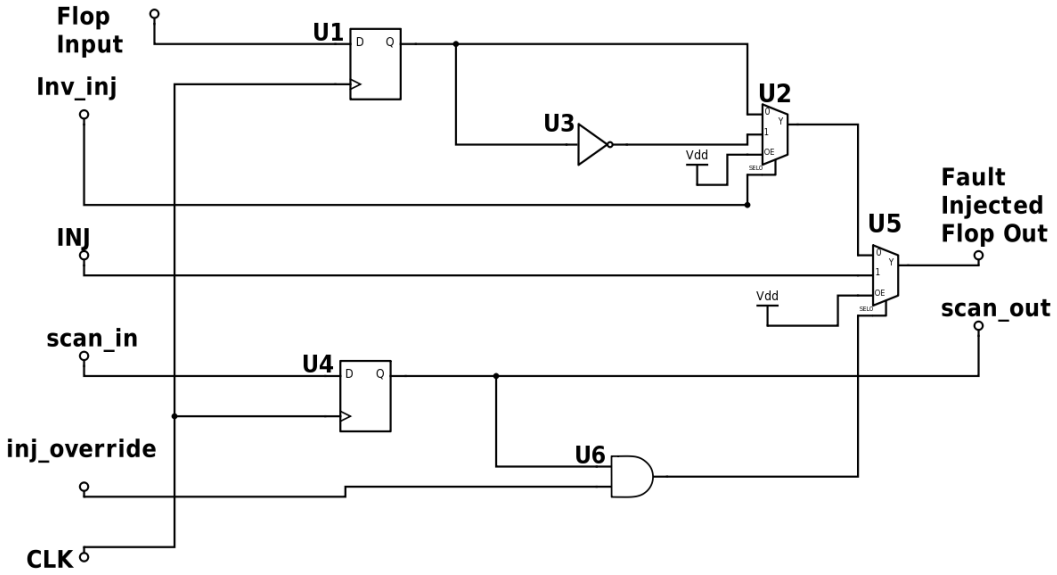


Figure 4.6: Modified Flops in ARM core

fault injection. After fault injection, the frequency is reduced to 30 MHz to avoid timing errors. To accommodate fault injection mechanism, each flop in the RTL of the ARM core is edited as shown in Figure 4.6. The original flop U1 is replicated into a copy U4 and stitched in a scan chain. The value in U4 along with *inj_override* determine whether fault is injected in output of U1. *INJ* injects stuck-at faults in the design and *Inv_inj* determines whether the output signal is to be flipped. The *inj_override* signal is to disable fault injection after specified number of cycles. The flop for fault injection is selected if *scan_in* is set to '1' for that flop. A system level fault controller, consisting of counters and a state machine for operating the scan, is responsible for handling the fault injection. To represent real scenarios of programs running on top of an operating system, a basic version of Linux (since ARM AMBER does not have virtual memory management, full version of Linux cannot be run) is used and programs are executed in this Linux environment. The fault controller specifies the following during injection:

1. Fault-injection type (stuck-at or transient)

2. Number of scan flops

3. Flop ID for injection

4. Time of fault injection

5. Fault injection control signals

Having defined the fault models, we now describe the test cases considered in this work for demonstration of detection performance.

4.3 Test Case 1: Inverted Pendulum System

4.3.1 System Description

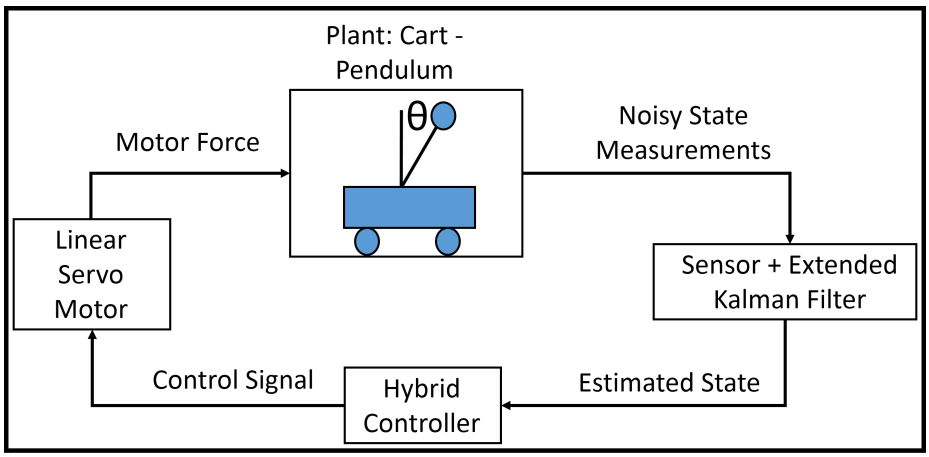


Figure 4.7: Inverted pendulum system mounted on a cart

As the first test case, the classical nonlinear control problem of cart-pole balancing is considered. Figure 5.9 shows the block diagram of the test case. The 'plant' consists of a single pendulum mounted on a movable horizontal cart and the control objective is to optimally move the cart such that the pendulum is balanced in an upright position. An extended Kalman filter (EKF) is used to predict unobserved states from noisy sensor measurements. A linear servo motor acts as the actuator to impart motion to the cart and a hybrid (with linear and nonlinear components) energy based controller is used. The differential equation governing the cart-pendulum system is given as,

$$\frac{d}{dt}\begin{bmatrix}\theta\\ \dot{\theta}\\ x\\ \dot{x}\end{bmatrix} = \begin{bmatrix}\dot{\theta}\\ f_{\theta 1}(\theta,\dot{\theta},u)\\ \dot{x}\\ f_{\theta 2}(\theta,\dot{\theta},u)\end{bmatrix} \tag{4.12}$$

where

$$f_{\theta 1}(\theta,\dot{\theta},u) = \frac{u\text{cos}(\theta) - (M+m)g\text{sin}(\theta) + ml\text{cos}(\theta)\text{sin}(\theta)\dot{\theta}^2}{ml\text{cos}^2(\theta) - (M+m)l}$$
$$f_{\theta 2}(\theta,\dot{\theta},u) = \frac{u + ml\text{sin}(\theta)\dot{\theta}^2 - mg\text{cos}(\theta)\text{sin}(\theta)}{M+m-m\text{cos}^2(\theta)}$$

and M is the mass of the cart, m is the mass of the pendulum's bob, θ is the angle between the pendulum and the vertical axis pointing upward, x is the position of the cart, l is the length of the pendulum (assumed mass-less), g is the acceleration due to gravity and u is the motor force applied to the cart to control its motion. The energy-based hybrid controller in [124] is implemented with the proposed detection scheme. Starting with the pendulum hanging vertically downwards the nonlinear energy-based controller is designed such that the mechanical energy of the pendulum-cart system is gradually increased by imparting energy at the correct instant to the pendulum. As the pendulum starts swinging higher and moves closer to the upright position, a negative state feedback controller takes over control to achieve the upright balance. The nonlinear control action of the energy-based controller, as derived in [124] is given as,

$$u_c = (M + m\text{sin}^2(\theta))\{g_1(x_d - x) - g_2\dot{x}\} + \tag{4.13}$$
$$mg\text{cos}(\theta)\text{sin}(\theta) + ml\dot{\theta}^2\text{sin}(\theta)$$

where the term x_d represents a sinusoidal reference input constructed from $(\theta, \dot{\theta})$ at each time instant and is designed to gradually increase the energy of the cart-balance system. g_1 and g_2 represent parameters of a second order transfer function from the reference input x_d to the actual cart position x. As the pendulum angle θ approaches the upright position, the linear negative state feedback controller takes over after $|\theta| \leq \theta_{th}$, where θ_{th} is the switching angle from one controller to the other. The linear control law is of the form

$$u_c = -\mathbf{K}.\mathbf{x} \tag{4.14}$$

where $\mathbf{K}$ is the feedback gain matrix and $\mathbf{x}$ is the system state vector. The gain $\mathbf{K}$ is determined from desired closed loop pole placements to meet specifications such as settling time and overshoot.

The angular position θ of the pendulum and the cart displacement x are measured as observations and modeled in the output vector $\mathbf{y} = g_c(\mathbf{x}, \eta)$ as

$$y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta} \\ x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} \eta_1 \\ \eta_2 \end{bmatrix} \tag{4.15}$$

where $\eta = [\eta_1 \quad \eta_2]^\intercal$ represent two Gaussian distributed measurement noise components. Equations (4.12-4.15) directly fit into the model described in (4.1-4.3). The linear servo motor used for position control applications in [125] is employed as the actuator generating the motion for the horizontal cart. The sensor, EKF, actuator and controller implementation for this test case are well studied topics in the literature and are not described in details here.

4.3.2 Simulation Results

All simulation experiments are conducted in Matlab on a Windows PC with 3.10 GHz processor. The pendulum-cart system is simulated with parameter values of M = 3 kg, m = 0.2 kg, l = 0.31 m and g = 9.81 m/s^2. The hybrid controller is designed such that the nonlinear controller switches over to the state feedback controller within a range of $\theta = \pm 30^\circ$ where $\theta = 0^\circ$ is the vertically upright position. For the linear feedback controller design, 20% maximum overshoot and a 2% settling time of 2 seconds are chosen. The resulting feedback gain $\mathbf{K}$ in (4.14) is determined using Ackermann's formula for pole placement. The measurement noise is modeled by a Gaussian distribution with mean 0 and standard deviation of 0.1. In order to compute the prediction functions, the cart-pendulum system is simulated with 100 different initial angles ranging between $\theta = 180^\circ$ and $\theta = 10^\circ$ over 20 seconds of simulation. To compute the optimal value of T_p, a 4th degree polynomial is selected to represent the nonlinear dynamics in (4.12) - thus ignoring the higher degree terms in the equivalent Taylor expansion of the sinusoidal dynamics in (4.12). The *state reduction factor* $f_{n \to p}$ is equal to 2 since two out of four system states are measured. From proposition I, this defines a minimum value $T_{p(min)} = 2 \times (4+1) = 10$. With an additional chosen $f_{noise} = 1.2$, the sliding temporal window length is fixed at $T_p = 12$. The unified variable $\mathbf{s}(t)$ in (4.4) is defined as $\mathbf{s} = \begin{bmatrix} y_1 & y_2 & u_c \end{bmatrix}^\intercal$. The function $F_m(.)$ is selected as a linear function as $F_c(\mathbf{s}) = y_1 + y_2 + u_c$. The objective is then to train the prediction function $F_c(.)$ such that the sum of the present unified variable $\mathbf{s}(t)$ is predicted from the past 12 observations. Two different choices of prediction functions are trained - i) a 3rd order truncated Volterra series by RLS adaptation with memory $M = 12$ and ii) a MARS model with 9 basis functions. After training the prediction functions, the error signal is generated as the difference between the two mapping function outputs. If the error signal lies above a certain threshold (pre-computed as $|e_{nom}| = 0.05$ for the noise statistic assumed), an error detection event is flagged.

Figure 4.8 shows a balancing operation where the inverted pendulum is hanging verti-

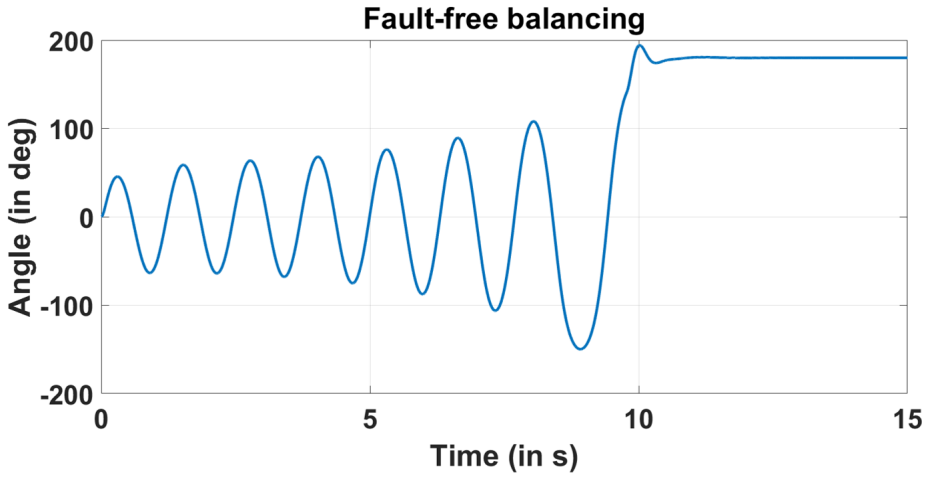


Figure 4.8: Plot of inverted pendulum angle with time in case of fault-free balancing

cally down initially with an angle of $\theta = 0°$. The nonlinear controller gradually increases the energy of the cart-pendulum system by periodically moving the cart in opposite direction resulting in gradual increase of the inverted pendulum angle towards the vertically upright position. As the pendulum approaches the vertically upright position, the effective control law is switched from the nonlinear control to negative feedback control for final balancing of the pendulum around $\theta = 150°$ (since the critical switching angle is $\pm 30°$). The switching of the controller from nonlinear to linear control is seen in Figure 4.9.

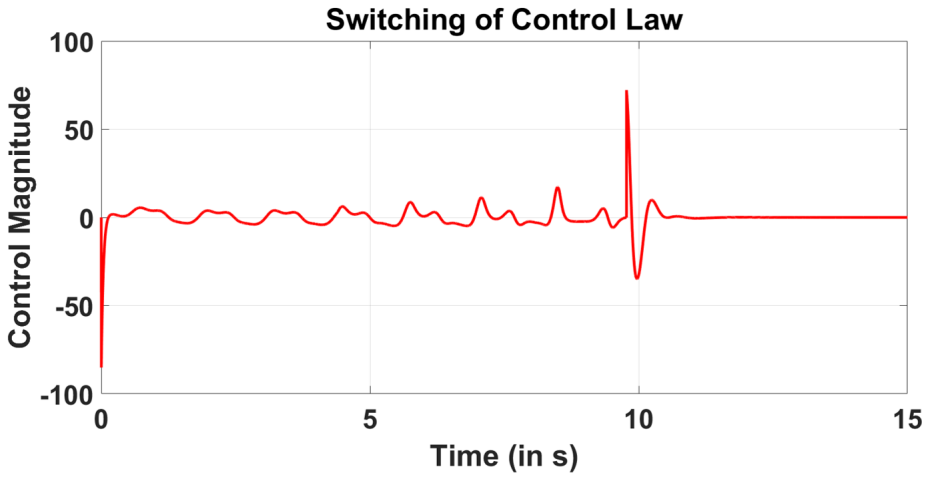


Figure 4.9: Switching of the nonlinear control to linear negative state feedback controller is demonstrated

The effectiveness of the proposed error detection scheme is heavily dependent on the

appropriate choice of the sliding window length T_p. If the appropriate choice of T_p is not adopted, the representative nonlinearity and memory effect of the system will not be sufficiently captured by the nonlinear mapping functions, resulting in higher magnitude of the error threshold e_{nom}. This is seen in Figure 4.10 where the MPCS error signal $e(t)$ for 7 different choices of T_p are plotted. It is seen that with increase of T_p from 1 to 12, the range of nominal excursion of the error signal decreases, thus reducing the error detection threshold e_{nom}. The optimum choice of T_p as mentioned in the previous section is 12 for this test case. However, if the choice of T_p is further increased, the error threshold doesn't reduce any further thus failing to provide any additional benefit albeit at a higher cost of computational effort and memory requirements.

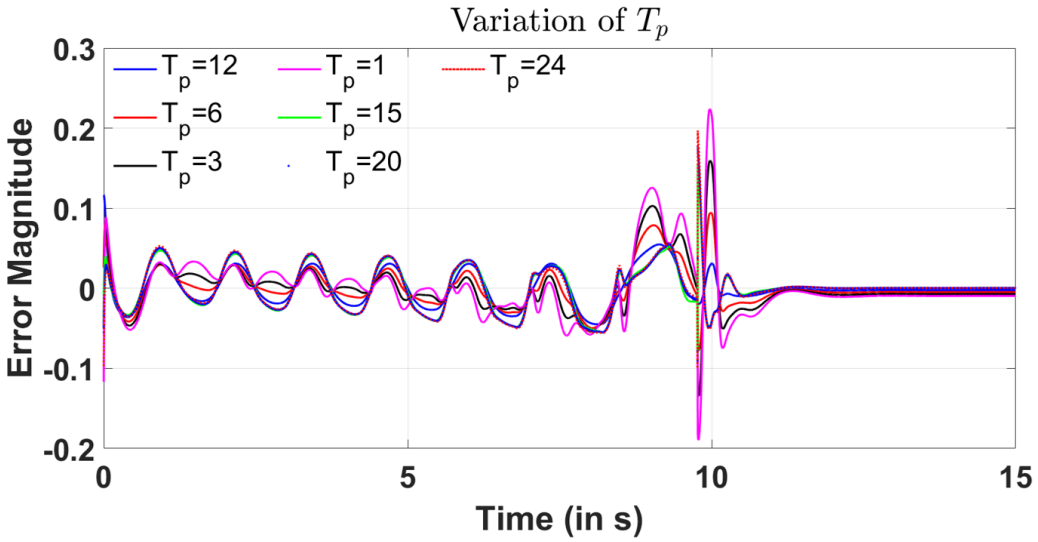


Figure 4.10: Plot of error signals with 7 different choices of temporal sliding window length T_p demonstrates the importance of selecting T_p properly

Figure 4.11 shows an illustrative scenario where soft errors, actuator disturbances and sensor perturbations are injected consecutively within a short time span for demonstration of detection performance. The y-axis in Figure 4.11 indicates the error magnitude value. The thresholding scheme performs limit value checking on the measurements and control signal for fault detection. The error signal in the thresholding scheme is generated by appropriate scaling of the vector $\mathbf{s}(t)$ for normalized comparing against the proposed scheme.

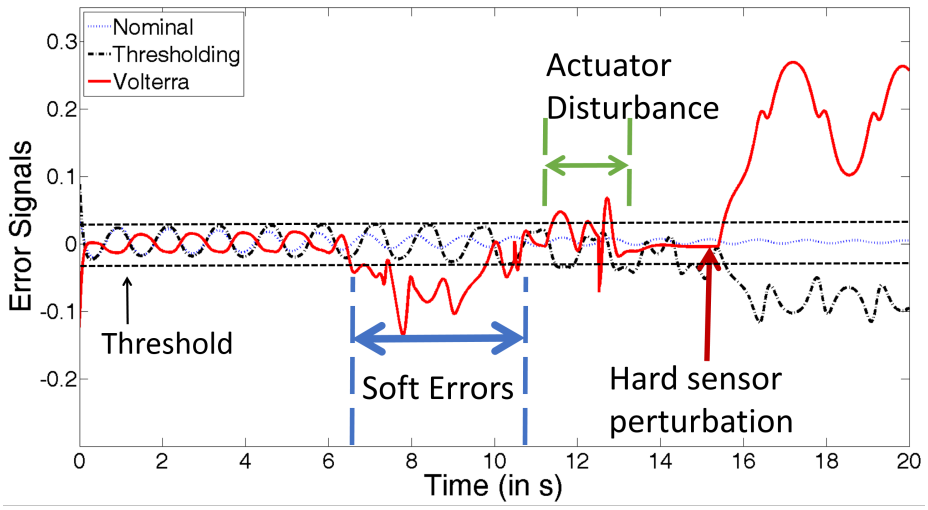


Figure 4.11: Error signal for proposed methodology using Volterra series as prediction functions (in solid red) and nominal fault-free response (dotted blue) is shown. The thresholding scheme is also depicted (in dotted black)

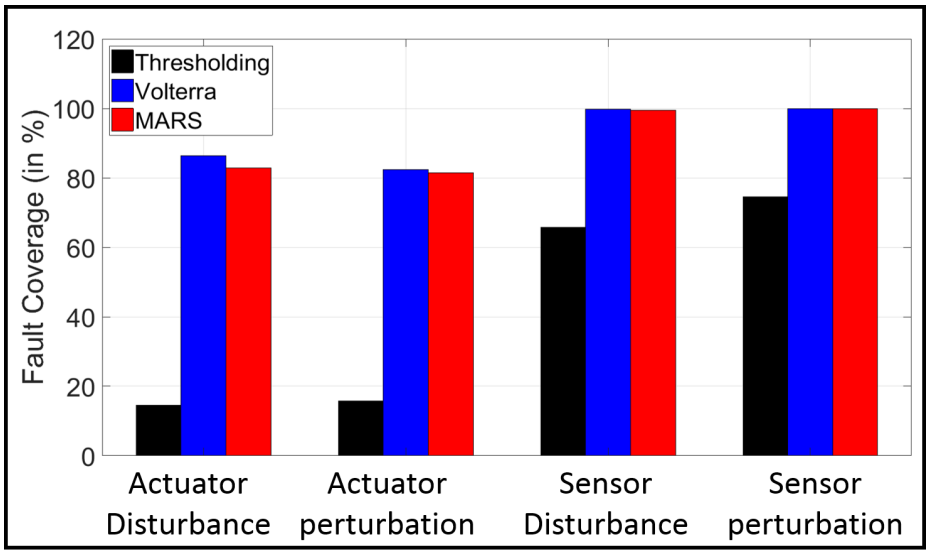


Figure 4.12: Comparison of fault coverage across sensor and actuator malfunctions

The cart position is constrained within a finite length platform, thus restricting the nominal range of x between $+5$ to -5. Similarly, the maximum control force that can be applied to the servo motor is restricted as $|u_c|_{max} = 3$. To map the nominal thresholding error within the same range of $|e_{nom}| = 0.05$, the thresholding error is generated as,

$$e_{th} = 0.005 \times x + 0.0083 \times u_c \tag{4.16}$$

The scaling in (4.16) generates a single error signal within the same threshold for the purpose of limit value checking. The arbitrary choice of scaling in (4.16) is selected only to enforce same threshold between the MPCS methodology and limit value checking for easy illustrative comparison. Between $t = 6.75s$ and $t = 10.8s$, soft errors in the form of single bit flips in the cache memories and physical register files are injected. The actuator disturbance is injected as 20% step decrease in the actuator output between $t = 11.2s$ and $t = 13s$. Finally, the hard sensor perturbation is injected at $t = 15s$ as scaling of the sensor values by a factor of 0.8, thus providing a 20% degradation. It is seen that sensor malfunctions create catastrophic effects and are detectable by both the methods - thresholding and mapped prediction. However, the soft computation errors and the actuator disturbance fail to generate perceptible changes in the system behavior that violate the established safety limits, thus evading detection by thresholding. The proposed mechanism quickly detect such errors and it is seen that the error signal crosses the nominal threshold. This is particularly helpful in early prognosis of system health before component degradation and aging effects cause noticeable change in system response. In all three fault injection methods, the error is instantaneously detected by the proposed approach, thus achieving low detection latency.

Figure 4.12 shows the fault coverage metrics for sensor and actuator fault injections. The fault coverage is defined as Coverage $= \frac{\text{\# of faults detected}}{\text{\# of faults injected}}$. A fault is detected if the error signal for that corresponding fault crosses the nominal threshold. The Volterra and the MARS based checker implements two Volterra kernel and MARS regression based schemes as illustrated in Figure 4.3 and compare the two outputs for the final error signal. For sensor and actuator disturbances, 100 step signals with step height distributed

uniformly between +20% to -20% of the actual signal are injected across the entire range of initial starting angles. Similarly, for parametric perturbations, 50 faults are created for both sensor and actuator with a Gaussian distributed parameter set of $3-\sigma$ equal to 25% of nominal value. Figure 4.12 illustrates that the proposed scheme with both Volterra kernel and MARS regression perform well in error detection with high fault coverage for all fault types. The proposed approach has 100% fault coverage for sensor faults and $\approx 80\%$ fault coverage for actuator faults. Actuator errors are typically difficult to detect and yet the mapped prediction checking has considerably higher fault coverage than the thresholding scheme.

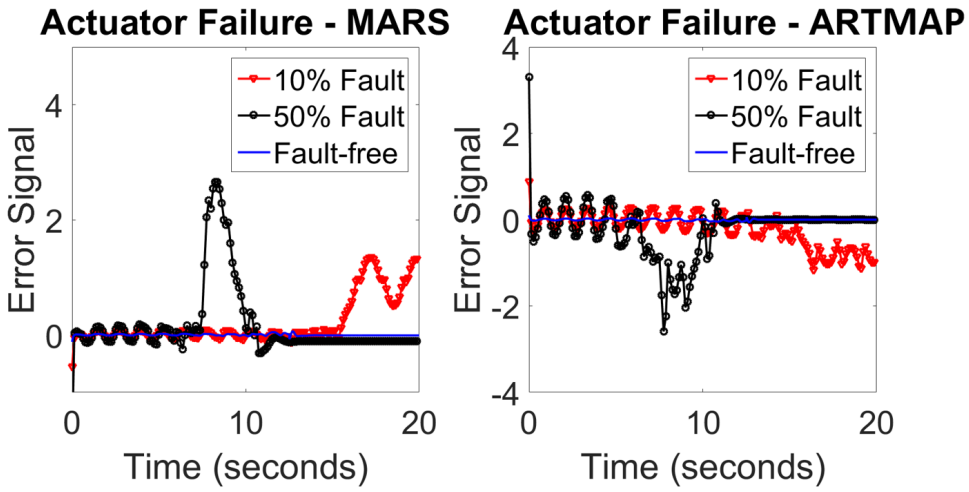


Figure 4.13: Error plots for 10% and 50% actuator faults with MARS and ARTMAP as prediction functions along with fault-free error signals

Figure 4.13 illustrates the detection of actuator errors using the MPCS approach with two different prediction functions - MARS and ARTMAP. Actuator faults are inserted as step disturbances with magnitude 10% and 50% of the nominal values. The two mapping functions are trained using nominal fault-free data from the 100 simulations mentioned previously. It is seen that both the mapping function choices are capable of detecting actuator faults. The 10% and 50% actuator faults are injected at $t = 15$ and $t = 7.5s$ such that the error signals are distinctly identified. It is seen that the 50% actuator fault generates higher error magnitude than the 10% fault which suggests that the error signal magnitude

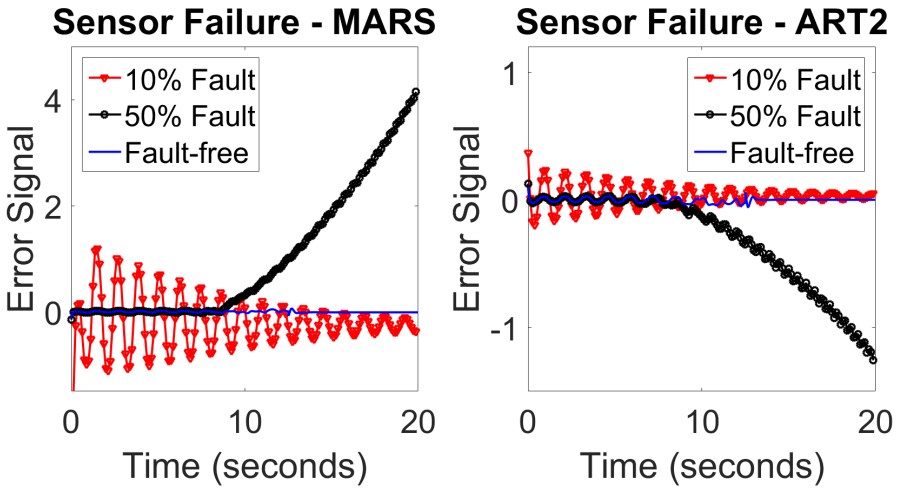


Figure 4.14: Error plots for 10% and 50% sensor faults with MARS and ARTMAP as prediction functions along with fault-free error signals

is indicative of the fault severity. The error detection in the presence of similar injected sensor faults are shown in Figure 4.14.

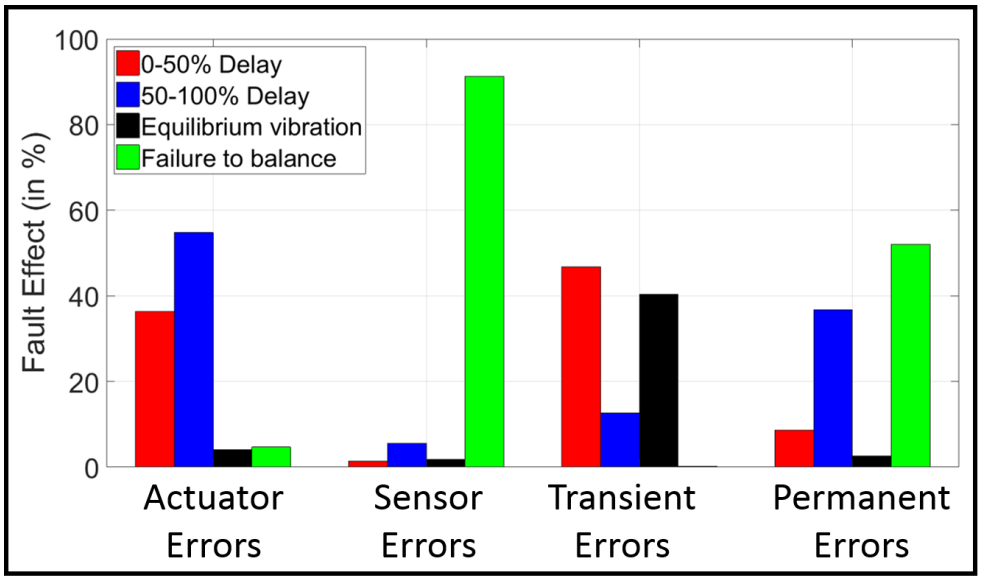


Figure 4.15: Application level fault effects due to different types of injected faults

Figure 4.15 demonstrates the functional behavior at the application level of pendulum balancing due to fault injection. Four different types of fault effects are classified - delay in balancing by 0-50% of the nominal time to balance, 50-100% delay, equilibrium vibration where the pendulum sways around the equilibrium position while attempting to balance

and failure to balance when the pendulum doesn't balance after twice the nominal time to balance. The equilibrium vibration fault effect is selected when the settling time specification is not satisfied. It is seen that sensor errors are most catastrophic since the pendulum fails to balance in 91% cases with injected sensor faults. Actuator errors mostly delay the balancing due to the imparting of incorrect actuation. Transient errors are rarely catastrophic and mostly delays the balancing or affects the balancing around the equilibrium position. Finally, injection of permanent errors (stuck-at faults at the micro-architecture level) fails to balance in 52% of the cases and delays the balancing in rest of the cases. The data reported on transient and permanent errors are from the subset of injected faults that cause SDC (Silent Data Corruption). The distribution of the different fault effects are tabulated in Table 4.3 and discussed later.

Table 4.2: Fault Coverage for Soft Errors in Simulation

Computational Data	**MPCS-Volterra**	**MPCS-MARS**
Computational Overhead	21%	19.5%
Memory Overhead	26%	22.3%
Number of bit errors injected	50000	50000
Transient bit error coverage	73.52%	66.29%
Permanent bit error coverage	98.95%	92.86%
Number of word errors injected	20000	20000
Transient word error coverage	94.54%	94.12%
Permanent word error coverage	99.52%	99.83%

Table 4.2 provides the implementation overhead along with the fault coverage metrics for different types of soft errors. Fault injection experiments are run 10 times with 5000 bit errors in each experiment. Similarly 2000 word errors are injected in each of 10 separate experiments. For bit errors, the microarchitectural block and the bit position of error

injection is chosen randomly. The instruction trace analysis from Pintool is exploited to carefully insert errors in the execution of controller and the checkers only. For word errors, 50% of the 32-bit word length are randomly selected for error injection. The computational and memory overhead is expressed as a proportion of the total cycles that represent the digital processor operation of control algorithms along with the checkers. This overhead varies with the controller choice and the reported data represents the overhead for the particular pendulum controller mentioned above. Among the 50000 bit errors injected, half of these errors are transient bit flips while the other half is permanent stuck-at faults (the same proportion is implemented for word errors). A 73% fault coverage implies that 27% of the injected fault effects are masked and don't cause any detectable data corruption. It is seen that transient word errors are easier to detect since multiple bit flips seldom result in masked fault effects. In the simulation framework, timeouts and crash are not explicitly detected by the transient error waveform. Such events are typically identified with a watchdog timer in actual systems. Thus the fault coverage reported in Table 4.2 is achieved by the proposed MPCS scheme along with an watchdog timer implementation.

Table 4.3: Fault effect distribution for different error types

Error Type	Fault Effect	MPCS-Volterra	MPCS-MARS
Transient bit errors	Masked	26.48%	25.91%
	SDC	68.46%	66.29%
	Timeout	3.69%	5.12%
	Crash	**1.37%**	2.68%
Permanent bit errors	Masked	1.05%	0.88%
	SDC	91.25%	92.86%
	Timeout	3.59%	3.14%
	Crash	4.11%	3.12%
Transient word errors	Masked	5.46%	5.88%
	SDC	74.21%	73.63%
	Timeout	4.58%	4.94%
	Crash	15.75%	15.55%
Permanent word errors	Masked	0.48%	0.17%
	SDC	37.45%	36.41%
	Timeout	1.28%	1.56%
	Crash	**60.79%**	61.86%

Table 4.3 shows the fault effect distribution due to injection of bit and word errors for the proposed scheme with Volterra kernel and MARS as prediction functions. From the emphasized data in Table 4.3, it is seen that only 1.37% of the injected transient bit errors result in a crash of program execution in the Gem5 simulator whereas 60.79% of the permanent word errors cause crash of the program execution. Permanent stuck-at faults in multiple bits of the microarchitectural blocks are catastrophic in nature and prevent normal program execution.

4.4 Test Case 2: Brake-by-Wire System

4.4.1 System Description

Modern vehicles equipped with antilock braking systems (ABS) are examples of brake-by-wire (BBW) adoption for improving the operational safety and system reliability. BBW systems integrate electromechanical actuators with electronic controllers and sensors. The primary objective of a BBW system is to limit the longitudinal slip in a braking event to prevent wheel lock-up conditions. A characteristic BBW system for a rear-wheel drive vehicle, as depicted in [126], is illustrated in Figure 5.19.

In a BBW system, the pressing of the brake pedal triggers the braking control algorithm implemented on an embedded microprocessor. Wheel-mounted sensors send speed and acceleration data to the electronic controller that generates the appropriate braking torque for application through electromagnetic brakes known as eddy-current brakes (ECBs). The control algorithm modulates the applied braking torque to extract maximum traction from the tire-road interface while avoiding wheel lock-up situations.

The BBW dynamics model described in [127] is utilized in this work. A straight-line braking event is assumed without any lateral motion. The longitudinal velocity equation and the rotational dynamics are:

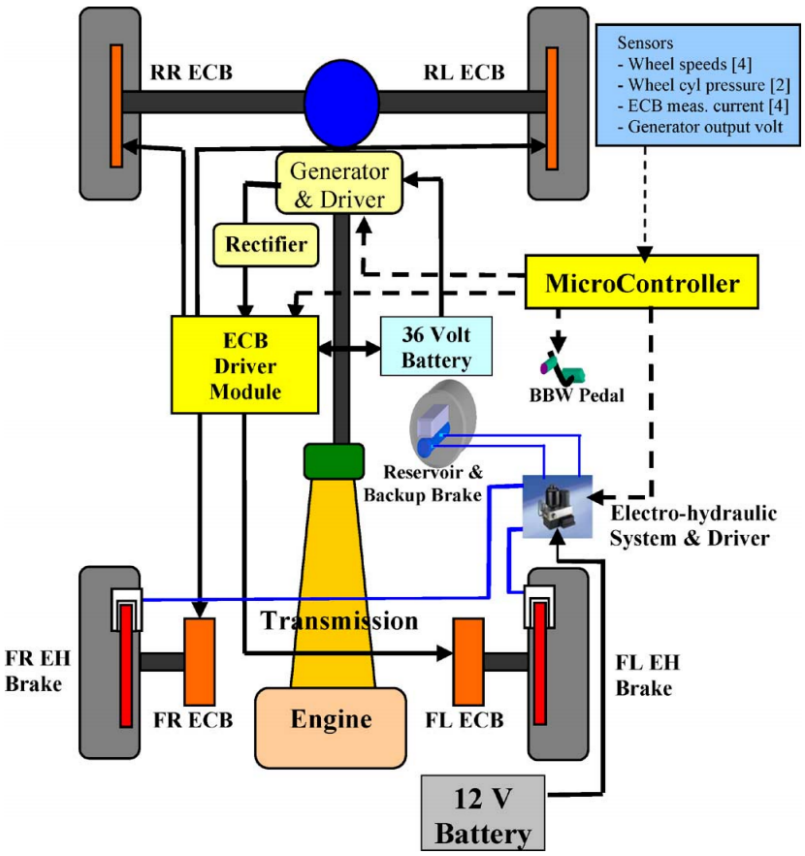


Figure 4.16: A brake-by-wire system for a rear-wheel drive vehicle

$$\dot{v}_x = -\frac{1}{M}\sum_{i=1}^{4}\mu_i(\kappa_i)F_{zi} \tag{4.17}$$

$$\dot{\omega}_i = \frac{1}{I_{wi}}(-T_{bi} + \mu_i(\kappa_i)F_{zi}R) \tag{4.18}$$

where v_x is vehicle longitudinal speed, M is total vehicle mass, ω_i is angular speed of ith wheel, I_{wi} is rotational inertia of ith wheel, R is the effective wheel rolling radius, F_{zi} is normal frictional force at the interface of ith wheel and road surface and T_{bi} is the generated brake torque at ith wheel. The tire-ground adhesion is determined by the wheel-slip ratio $\kappa_i(t)$ and defined as:

$$\kappa_i(t) = \frac{v_x - R\omega_i}{v_x} \tag{4.19}$$

The nonlinear function $\mu_i(\kappa)$ in (4.18) defining the friction coefficient as a function of time-varying slip ratio $\kappa_i(t)$ represents the nonlinearity in the BBW system. This function depends on surfaces (asphalt, snow, ice, etc.) and the friction coefficient curves vary significantly across slip-ratio. In general, the maximum braking force is generated at the peak friction coefficient for which the slip-ratio value ranges between 0.1-0.2. Equations (4.17-4.19) represent the plant equations of the BBW test case.

Wheel lock-up is a condition in which one or more wheels start having higher rate of angular deceleration than the equivalent linear deceleration of the vehicle, causing loss of traction, slippage of wheel over road surface and eventually loss of vehicle control. The primary objective of a BBW controller is to avoid such a situation by modulating the applied braking torque T_b while maintaining the critical slip ratio. In this work, a sliding mode nonlinear controller for ABS is implemented as derived in [126]. The ABS control law is expressed as,

$$\begin{aligned} T_{bi} = R\alpha_{si}\kappa_i F_{zi} + \frac{I_{wi}}{v_x}\frac{\omega_i}{M}\sum_{i=1}^{4}\alpha_{si}\kappa_i F_{zi} \\ + \eta\frac{I_{wi}}{R}v_x * \text{SAT}(\frac{\kappa_{th} - \kappa_i}{\phi}) \end{aligned} \tag{4.20}$$

where the saturation function $\text{SAT}(\frac{x}{y})$ is defined as,

$$\text{SAT}(\frac{x}{y}) = \begin{cases} \frac{x}{y} & \text{if } |\frac{x}{y}| \leq 1 \\ \text{sgn}(\frac{x}{y}) & \text{if } |\frac{x}{y}| > 1 \end{cases} \tag{4.21}$$

The terms α_{si}, κ_{th}, η and ϕ are parameters of the sliding mode controller as defined in [126]. A BBW-system operates completely in highly nonlinear mode and linearization is not feasible around any particular equivalent point.

4.4.2 Simulation Results

The BBW test case is simulated with parameter values of $M = 1300$ kg, $R = 0.3$ m and $I_w = 0.6$ kg/m^2. The friction-slip curve for asphalt surface mentioned in [126] is utilized. Similar to the cart-pendulum test case, an equivalent 4th degree polynomial is chosen to model the system nonlinearity, thus defining a minimum value $T_{p(min)} = (4 + 1) = 5$ (the *state reduction factor* is 1 in this case since all states are assumed to be measured). With the additional chosen noise corruption factor $f_{noise} = 1.2$, the sliding temporal length is selected as $T_p = 5 \times 1.2 = 6$. A 3rd-order truncated Volterra kernel is chosen as the prediction function F_c and trained with braking data for 1000 different initial vehicle speeds ranging from 50 mph to 100 mph. The function F_m is selected to generate the sum of wheel-speeds. Gaussian distributed measurement noise with mean 0 and standard deviation of 2 rad/s is assumed while measuring angular speed ω_i of each wheel. Figure 4.17 illustrates the catastrophic effect of a soft error in the microcontroller of the BBW system. Figure 4.17.(a) and (b) plot the angular speed of the left front wheel and the left rear wheel respectively. The initial vehicle longitudinal speed is 60 mph with a uniform angular speed of 88 rad/s across all four wheels, providing a slip ratio almost equal to 0 and thus maintaining full contact with the road surface. At time $t = 1$s, full braking is initiated for rapid deceleration of the vehicle resulting in uniform reduction of angular speed in both front and rear wheels. A spurious bit flip is injected in the issue queue at $t = 3$s, resulting in incorrect computation of the ABS control law. This causes a temporary wheel lock-up of the front wheel and drastically reduces the angular speed thus losing traction. As a result, the linear velocity suddenly increases due to wheel slippage as evident in Figure 4.17.(b) indicating a catastrophic situation. However, the robustness of the sliding mode

ABS controller and the transient nature of the underlying error provides quick recovery of traction and produces the appropriate retardation.

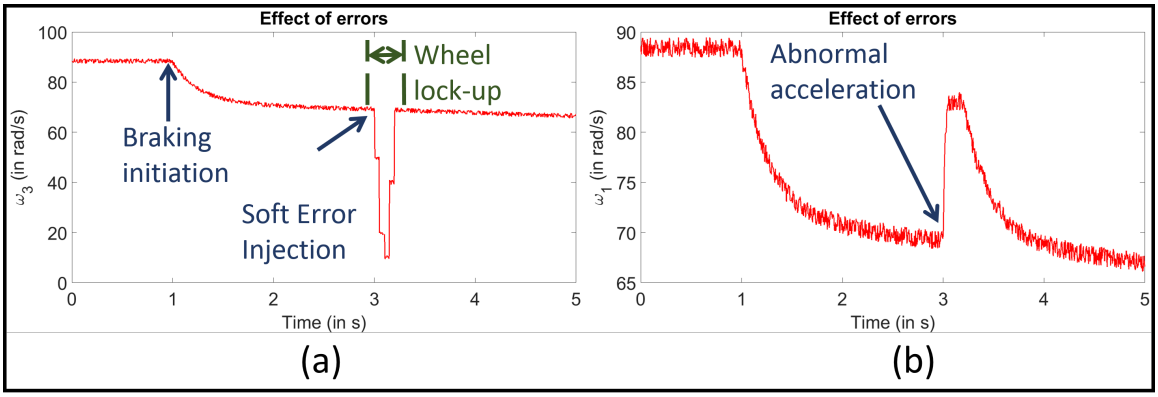


Figure 4.17: Plot of angular speed for a) rear wheel and b) front wheel for a braking event operating with a soft error

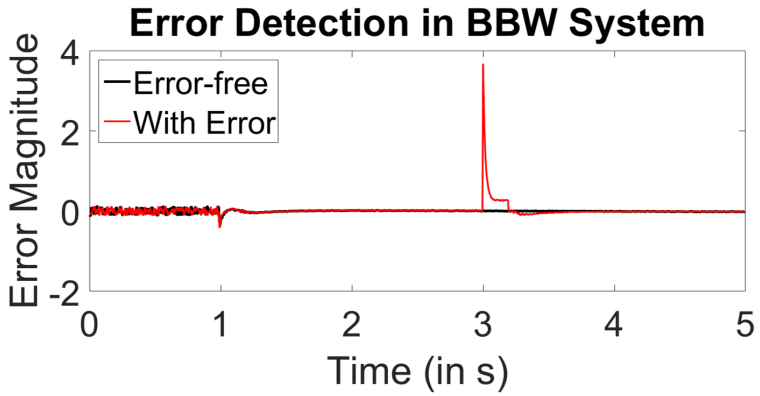


Figure 4.18: Plot of error signal without errors and with soft computation errors

Figure 4.18 demonstrates the detection of the bit flip error with the MPCS-Volterra scheme. The error signal clearly shows a prominent signature different from the nominal error signal without any fault injection. The injected error is instantaneously detected with low latency. For statistical fault injection experiments, 500 faults are injected by perturbing the wheel sensor readings and the actuator (ECB) signals with step disturbances uniformly distributed between +20% and -20% of the nominal signal values at different time instants. 10 fault injection experiments are executed with 5000 bit errors and 5000 word errors in each fault injection experiment.

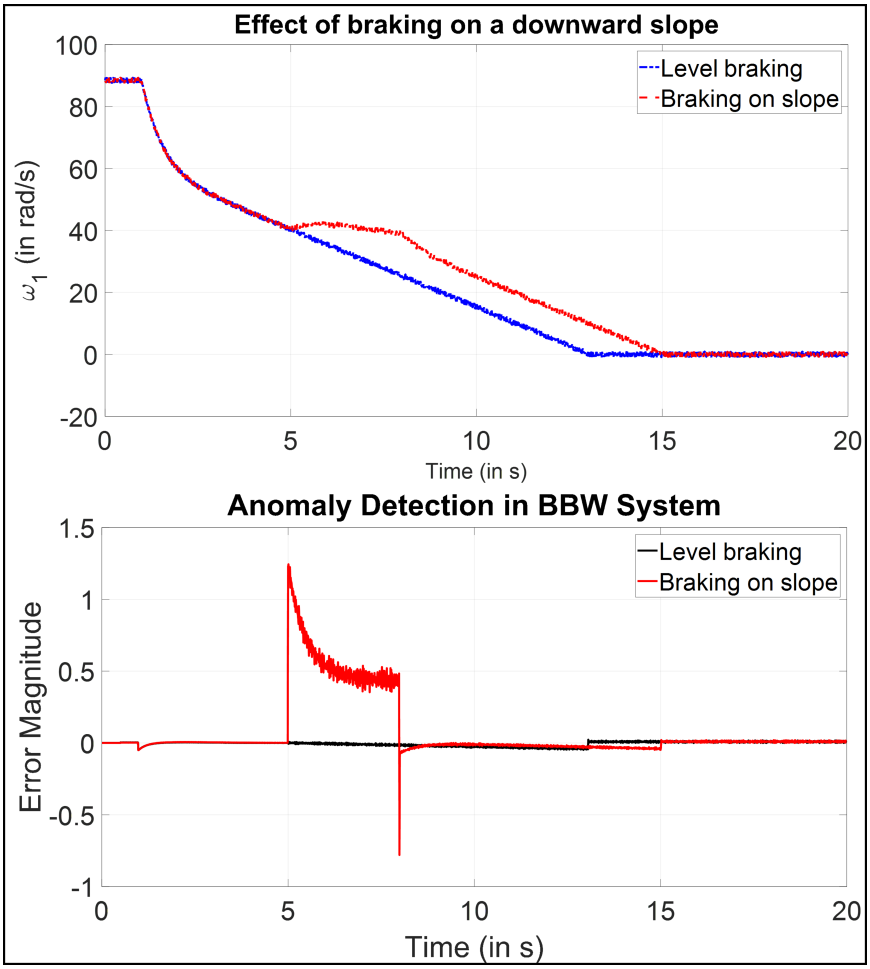


Figure 4.19: Braking on a slope indicates a scenario for which the mapping functions are not trained. It is clearly seen that due to change of system dynamics on a downward slope, the mapping function is unable to predict the response trajectory, triggering the detection of an environmental anomaly

Figure 4.19 illustrates an environmental anomaly situation as discussed in Section 4.2.1. Similar to Figure 4.17, braking is initiated on a level road at $t = 1s$ from a wheel rotation of 88 rad/s. At $t = 5s$, the road starts sloping downward at an angle of $5°$. This is simulated by adding an extra weight component of $Mg\sin(\theta)$ in (4.17) where θ is the downward slope angle. The increased downward force slows down the braking effect at $t = 5s$ as seen in Figure 4.19. The slope ends at $t = 8s$ and the same braking trajectory is restored henceforth. The prediction functions are trained using braking data on a level slope as mentioned previously and cannot predict the system dynamics on a downward slope. The

bottom subfigure demonstrates detection of such a scenario for which the MPCS is not trained. This indicates that the proposed approach is not only suitable for fault detection but is also capable of identifying environmental aberrations that change the plant dynamics due to situational conditions.

Table 4.4 shows the fault coverage for the BBW system operating under different faults and the computational overhead of the implemented scheme. Similar to the first test case, the fault coverage is 100% on sensor errors as all injected faults are detected. The coverage for soft errors and actuator faults is also high as shown in the Table. Compared to the previous test case, the computational and memory overhead is lower since the controller overhead is higher in this case.

Table 4.4: Fault coverage and computation overhead for MPCS-Volterra in BBW system

Computational Overhead	12% (92998445 on 774987049 cycles)
Memory Overhead	18.6%
Fault Coverage on bit errors	88.64% (44321 out of 50000 injected errors)
Fault Coverage on word errors	98.51% (49256 out of 50000 injected errors)
Fault Coverage on sensor faults	100% (250 out of 250 injected faults)
Fault Coverage on actuator faults	96.4% (241 out of 250 detectable faults)

4.5 Hardware Results

4.5.1 Fault Injection in FPGA Testbed

The hardware experiments are performed on the Xilinx Spartan-6 FPGA SP605 board shown in Figure 4.20. As described in Section 4.2.3, the 3012 flip-flops are modified for

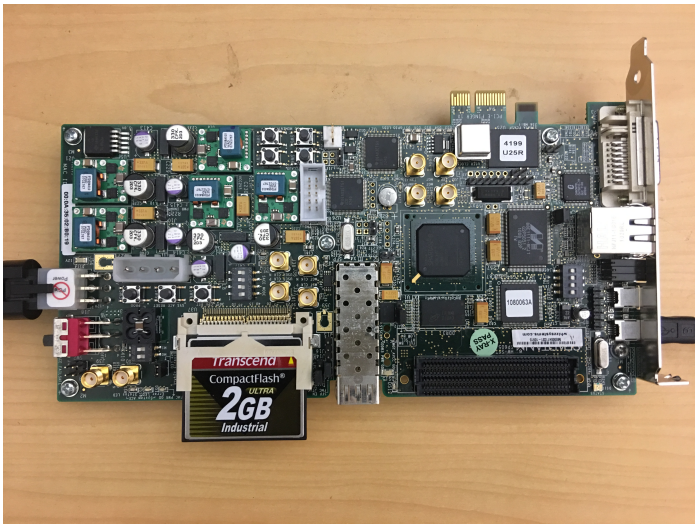


Figure 4.20: Xilinx Spartan-6 SP605 FPGA board for fault injection experiments

fault injection capabilities. After performing the RTL modifications, the design is synthesized and mapped to the Configurable Logic Blocks (CLBs) present in the FPGA. Each CLB consists of 4 to 6 inputs, multiplexers and flip-flops. The muxes used in the modified flip-flop in Figure 4.6 are already available in the CLBs and do not cost additional overhead. Two ChipScope cores ICON and VIO are generated by Xilinx Coregen and inserted in the design for fault control purposes. The VIO (Virtual Input/Output) core is used to provide the command for system level fault control. The ICON (Integrated Controller) provides communication between VIO and the host computer. For bit errors, the system clock is gated off when the fault sequence is scanned in. The system clock is activated in the immediate cycle following the bit flip. The entire system with the fault injected core, fault controller cores and peripherals is synthesized in a bit file and downloaded to the FPGA for fault injection experiments. A complete fault injection run is illustrated in Figure 4.21.

Each fault injection run involves programming the FPGA with the RTL-modified processor and then issuing appropriate ChipScope commands as shown in Figure 4.21. The flop ID and injection time are randomly chosen for fault injection. A basic Linux operating system is next booted and the application under fault is loaded through the UART serial interface from the host PC. After execution of the application on the fault-injected processor

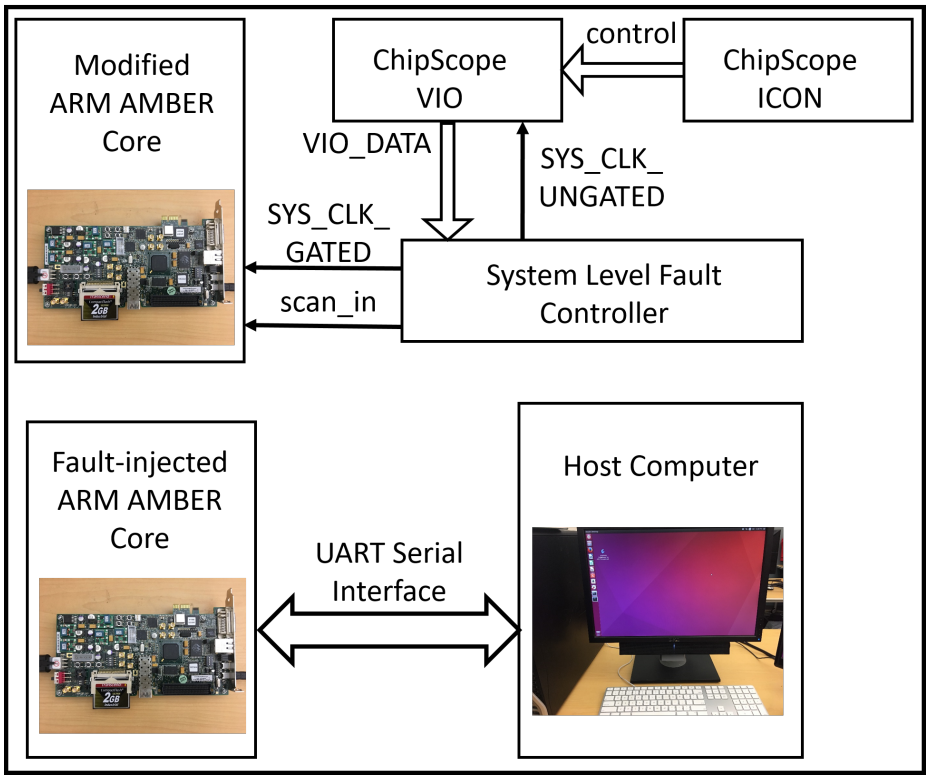


Figure 4.21: Test flow for a complete fault injection run involves the introduction of faults from the system level fault controller and then booting a Linux kernel and executing an application over UART serial interface

core, data is collected back for post-processing and fault effect analysis.

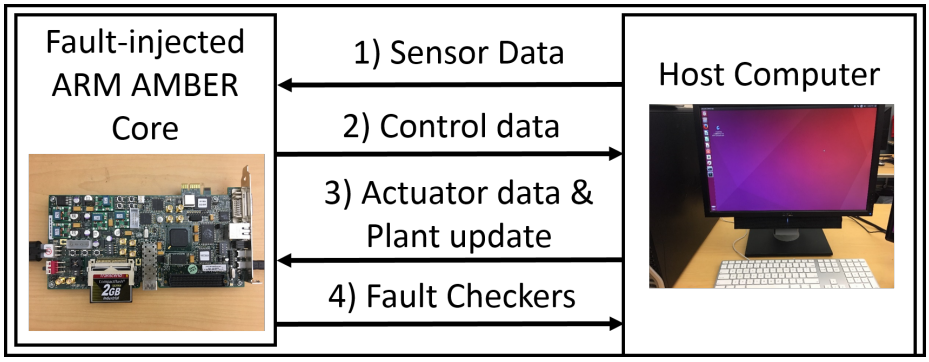


Figure 4.22: Execution of fault injection on control application test cases

Figure 4.22 shows the execution flow for fault injection on the test cases mentioned previously. As discussed in Section 4.2.3, only the control algorithm and fault checkers are implemented on a digital processor core. If the entire control program is executed on the faulty core as shown in Figure 4.21, soft errors affect the operation of sensors, actuators

and plant that does not mimic practical scenarios. Hence, the execution flow of Figure 4.22 is adopted, where each sampling time iteration of the test case is simulated with a bidirectional data exchange between the host computer and the fault-injected ARM core. At iteration 1, the plant is initialized and sensor model is simulated on the host PC and the control algorithm with measured data as inputs is sent as a standalone application to be executed on the FPGA core. The output from the controller is sent back to the host PC where the actuator model and plant equations are simulated. Finally the fault checker algorithm with the present and past measurements as input data is executed on the FPGA core. This entire process is repeated over iterations to complete a full simulation of a test case.

One program execution on the FPGA takes 80 seconds - the total time of injecting the bit-flip in the ARM core, booting Linux on the faulty core and application execution. Each of the temporal iterations of a control problem requires two FPGA executions as shown in Figure 4.22 - one for the controller and another for the fault checkers. Thus the simulation of 1 time sample of a control program requires 178 seconds - 160 seconds for FPGA runs and 18 seconds for data exchange and PC simulation. The brake-by-wire test case is simulated for a duration of 10 seconds with sampling time of 0.25 seconds generating 40 temporal iterations thus requiring 40 injection runs on the FPGA. Hence each experiment of the test case requires $40 \times 178s \approx 2$ hours. In the FPGA execution of each temporal iteration, a random flop is selected and a random bit flip is injected. Both the control algorithm and the checker at each iteration runs on the same bit-flip injected core. At the next iteration, a different soft error is injected and the process is repeated for the entire 40 temporal iterations. After each iteration, the fault effect of the injected bit-flip is classified as one of the four outcomes mentioned in Section 4.2.3 - masked if there is no corruption of program data from fault-free values, SDC if the fault checker value crosses the detection threshold, timeout if the program doesn't complete even after running 5 times longer than fault-free case and crash if Linux fails to load or program execution is fatally

terminated. A total of 100 such experiments are performed over a time-span of 9 days. The fault outcome distribution of the 100 experiments is shown in Figure 4.23.

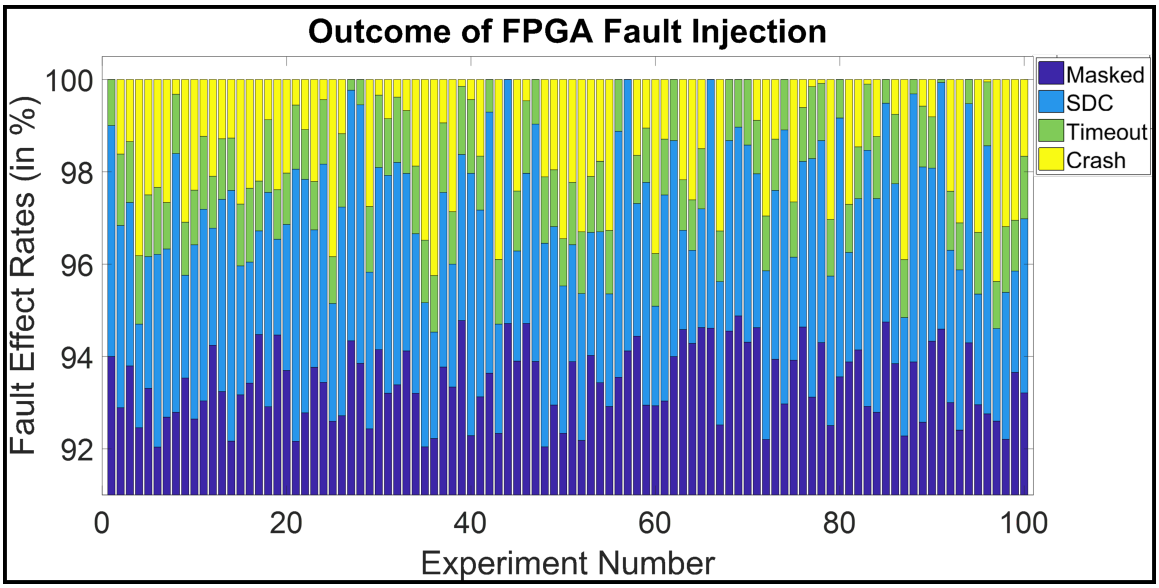


Figure 4.23: Distribution of fault effects for 100 experiments of the brake-by-wire test case on the FPGA

On average 93% of the injected bit flips are masked as these do not cause any perceptible corruption of the program data from stored fault-free values. Hence, the y-axis that plots the fault effect rates start from 92% for aid of visual comparison. 4.5% of the injected bit flips corrupt the controller algorithm or the fault checker evident from the error value crossing the established threshold. The mean rates of timeout and crash are 0.2% and 2.3% respectively. In a practical situation, soft errors do not strike at each time instant. However, the primary reason that bit errors are injected at each iteration is explained from the masked outcome rate in Figure 4.23. Unlike the Gem5 based fault injection experiments, no trace analysis is available before fault injection. In addition, the mapping between individual FPGA flip-flop and the high level operation it performs in the program execution including running the operating system, is unknown. Thus, a bit error injection to target the specific instruction of program execution is not possible. The masked effect is created when a bit error affects a particular flip-flop in a way that does not affect the program execution in any manner. The bit errors that cause crash and timeout affect the operating system or crucial

system routines. Hence, to ensure an observable corruption of system data, bit flips are injected at each time instant.

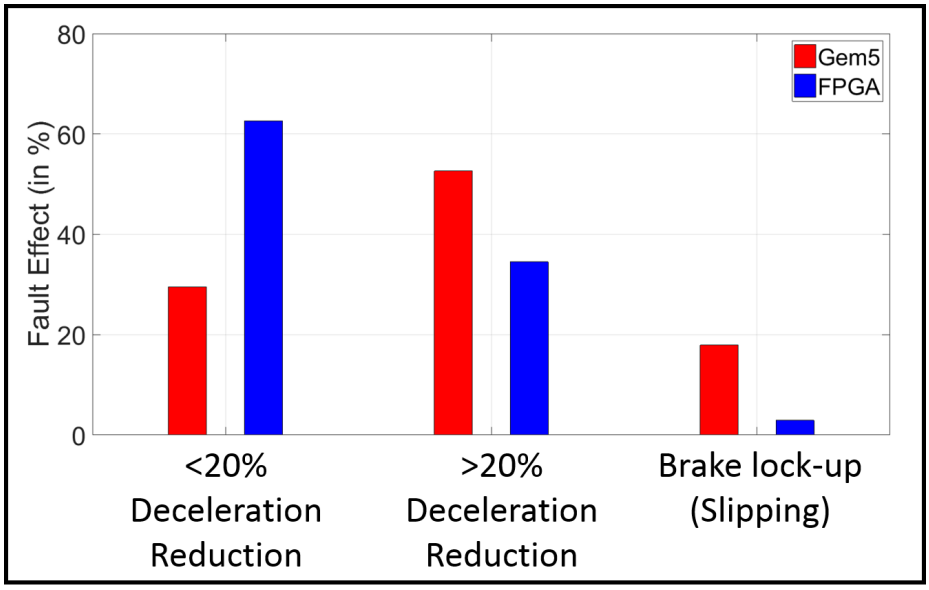


Figure 4.24: Comparison of application level effects between Gem5 fault injection and FPGA-based flop level error insertion

Figure 4.24 illustrates the application level effects of transient bit error injections for the BBW system and draws a comparison between Gem5-based fault injection scheme and the FPGA-based flop error insertion mechanism. Three different classes of system level fault effects are defined - i) $< 20\%$ reduction in deceleration, ii) $> 20\%$ reduction in deceleration, and iii) brake lock-up. Injection of transient bit errors disrupts the correct computation of braking torque by the sliding mode controller. A reduction in the applied braking torque from the required value decreases the effective deceleration and fails to slow down the vehicle sufficiently whereas an increase in the applied torque raises the slip ratio inadvertently, causing wheel lock-up as shown in Figure 4.17. The data for the Gem5 plot is selected from a subset (17598 out of 44321 detected bit errors) of the transient errors reported in Table 4.4. The data for the hardware fault injection scheme is selected from the errors that cause SDC shown in Figure 4.23. The critical percentage of deceleration reduction of 20% is chosen based on simulation data where it is observed that if the deceleration decreases

by more than 20%, the stopping distance criterion is not met. Figure 4.24 shows that due to targeted fault injection in Gem5 based scheme, the fault effects are more catastrophic - 52% cases not satisfying stopping distance criterion and 18% causing wheel lock-up. With the fault-injected ARM core, 63% of the cases are non-catastrophic with brake lock-up occurring in only 2.9% of the cases.

4.5.2 Self-Balancing Robot

Description

To demonstrate the error detection performance on an actual system, the Balboa 32U4 [128] self-balancing robot is chosen. The controller, actuators and sensors are described as:

i) Controller - The control board of the robot is integrated around a Atmel microcontroller, clocked by a 16 MHz crystal oscillator along with an Arduino-enabled bootloader that makes the robot highly programmable.

ii) Actuator - Two on-board Texas Instruments DRV8838 motor drivers power two micro-metal gear motors driving each of the two wheels separately. There are 4 actuator signals used to control the motor drivers - rotational direction and speed for each motor. The motor terminals are controlled by pulse width modulated (PWM) signals from the drivers. Each of the two motors is a 6V, brushed DC motor with customizable gearbox (a 51.45:1 gear ratio is chosen in the setup). The motors have a no-load speed of 625 RPM and 120 mA current at 6V. Each of the motors drives a wheel with 80 mm diameter.

iii) Sensor - Each drive motor on the robot has a quadrature encoder system consisting of a magnetic disc attached to the extended motor shaft and a pair of Hall effect sensors mounted on the control board. These encoders are used to track the rotational speed and direction of the robot's wheels. In addition, there are two on-board inertial sensor chips that assist the robot to determine its own orientation by implementing an inertial measurement unit (IMU). The first chip (ST LSM6DS33) combines a 3-axis digital linear accelerometer

and 3-axis gyroscope into a single package. It provides 6 independent acceleration and angular rate readings as a 6 DOF (degree of freedom) data. The second chip (ST LIS3MDL) is a 3-axis digital magnetometer that provides the robot a sense of location with respect to the magnetic field. These constitute 13 sensor measurements available for the controller - 4 quadrature encoder readings and 9 inertial measurements.

Balancing algorithm

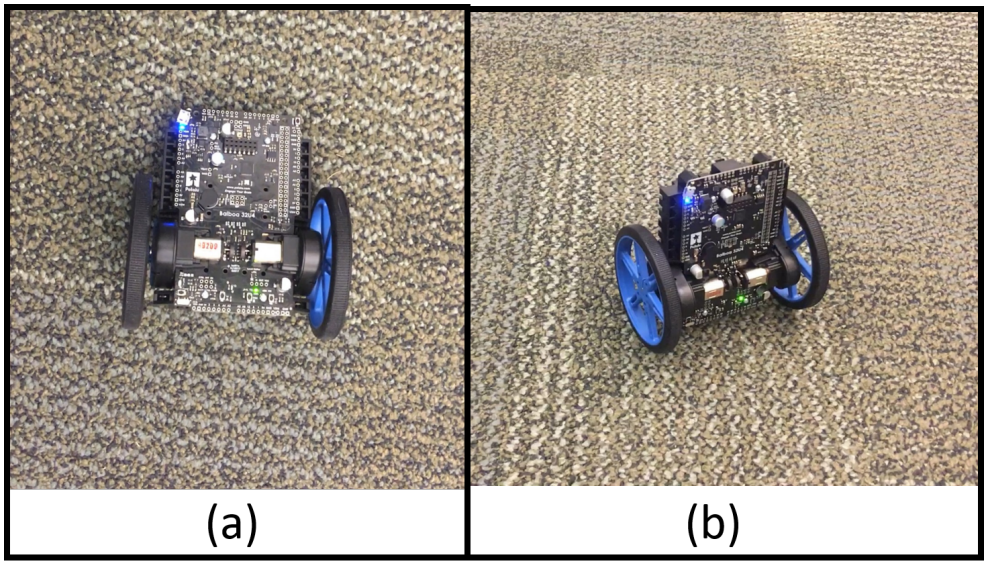


Figure 4.25: Self-balancing Balboa robot

The self-balancing robot is shown in Figure 4.25.(a) in the lying-down position and in 4.25.(b) in the vertically upright position through dynamic balancing. Similar to the inverted pendulum test case, a hybrid controller combining both linear and nonlinear control laws is used for balancing the robot. A nonlinear controller is used to make the robot get up from a lying-down to a near-vertical position by providing quick maximum motor speed signals in alternate directions. A phase space plot of the balancing algorithm is shown in Figure 4.26.

The two axes in Figure 4.26 plot the angle of the robot with the vertical axis and the angular rate of rotation. The vertically upright position is represented as the origin of the

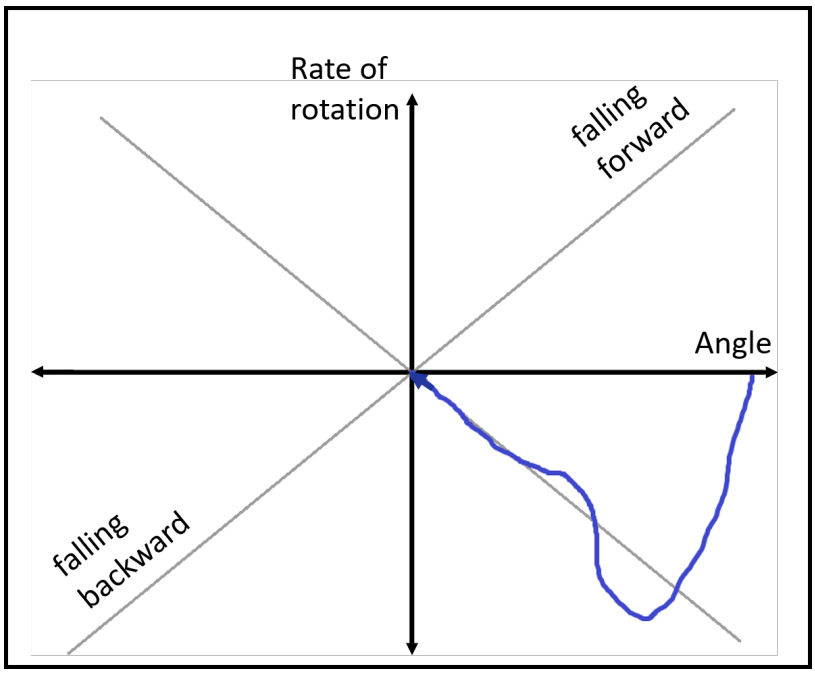


Figure 4.26: Phase plot of the balancing algorithm shows the balancing trajectory implemented by the hybrid controller

phase space with angle and angular rate equal to zero. One of the diagonal trajectories with +45° angle to the horizontal axis is unstable and indicate the falling of the robot, either forward or backward. The robot is falling forward when both the angle and angular rate of rotation is positive and increasing. Similarly when both angle and angular rate is increasingly negative, the robot is falling backward. The second diagonal with -45° with the horizontal represents the desired trajectory for balancing where the angle and angular rate is opposite in sign. The balancing algorithm starts with the robot lying down with an angle of 110° and zero angular rate. The nonlinear controller aims to bring the robot close to the balancing trajectory by fast switching of forward and backward motor speeds. As shown in the phase plot, the nonlinear controller switches to a PID controller once the robot's orientation crosses the balancing trajectory. The PID controller generates the required motor drive to reduce the error between the robot's actual orientation and the balancing trajectory and in the process, balances the robot in the vertically upright position.

Results

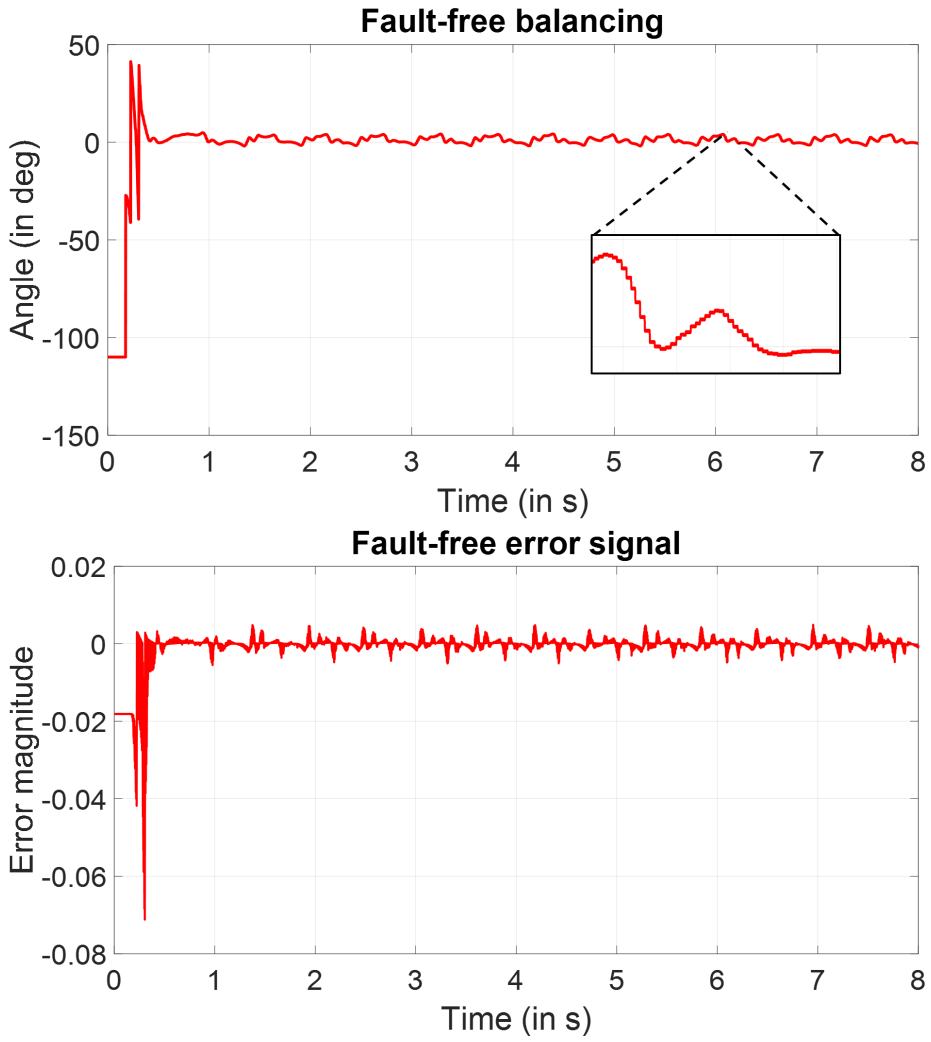


Figure 4.27: Angular orientation of the robot along with the MPCS error signal without any fault injection

To compute the prediction function F_c, 100 experiments of the robot balancing are performed and temporal data from the sensors and the motor drive speed are recorded. Since the Atmel micro-controller has only 32 KB of flash memory, temporal data of $t = 8$s can be recorded at a stretch. So, 8s time snippets of data are recorded for 100 experiments to build the mapping function F_c. The balancing algorithm operates on 10 sensor measurements - 6 DOF data from the accelerometer and the gyroscope and 4 quadrature encoder readings. The two motor drive currents form the actuation signals. The sampling time of

the data is 1ms. The sliding window length is selected as $T_p = 6$ samples. The function F_m is chosen to form the normalized sum of 12 signals - 10 sensor measurements and 2 actuation currents at each instant. A MARS model is built to predict this value from the past data over 6 samples. A nominal balancing experiment with the fault-free error signal is shown in Figure 4.27.

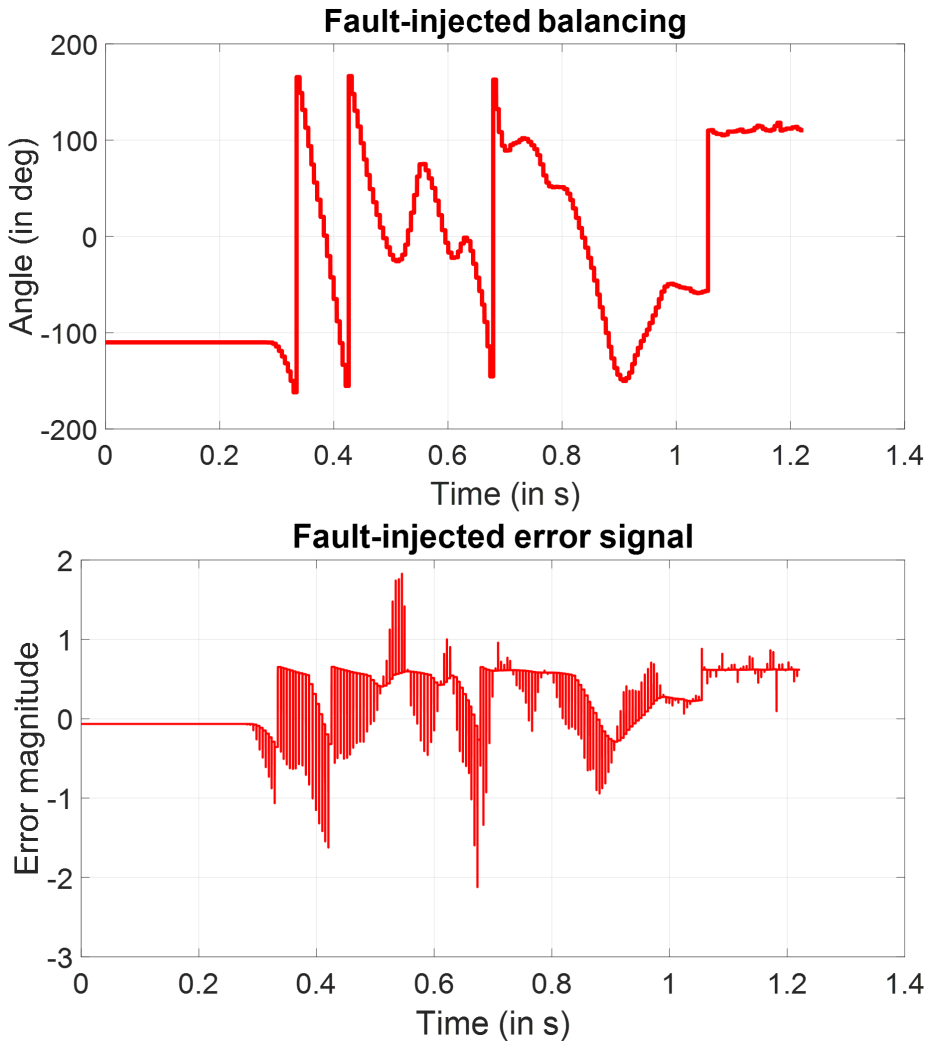


Figure 4.28: Angular orientation of the robot along with the MPCS error signal without any fault injection

Figure 4.27 shows the angular orientation of the balancing robot in the top subfigure. The robot starts from a lying-down position of 110° and the nonlinear controller swings up the robot for balancing and the angle of the robot stays within $\pm 5°$ while it is balanced

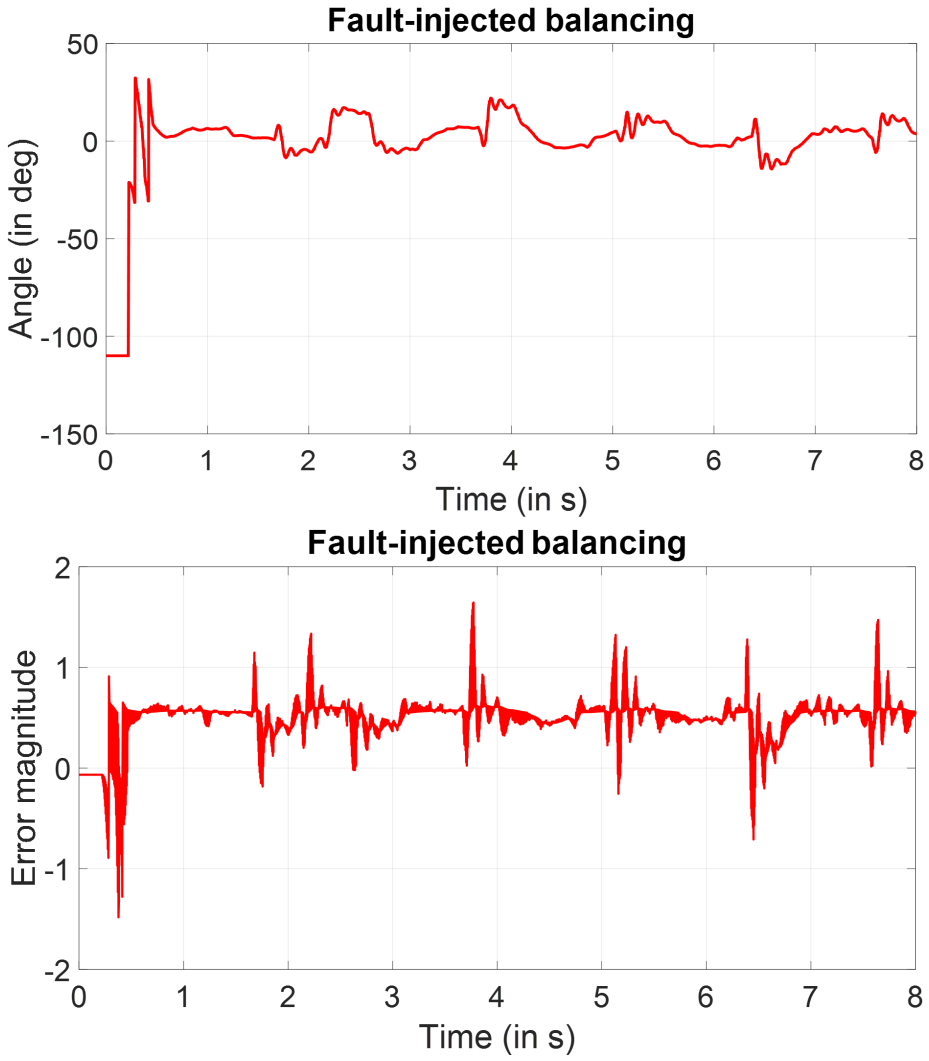


Figure 4.29: With a sensor fault injected, the robot swings around the equilibrium and fails to perfectly balance itself producing a high error signal

upright. The step-like response shown in the inset illustrates the PWM drive of the motors. In the bottom subfigure, it is seen that the error signal is close to 0 for a nominal balancing indicating the absence of any faults and anomalies. The error detection threshold is chosen as $e_{th} = 0.1$ from the nominal data collected from 100 experiments.

Figure 4.28 shows the response of the robot while attempting to balance with a sensor error of 20%. The accelerometer reading from the sensor is altered artificially with a bias of 20%. The angle and the MPCS error signal are shown where it is seen that as the robot attempts to stand up from the lying down position, it vibrates around the vertical equilib-

rium position of $\theta = 0^{\circ}$ and sways the other way. The robot makes multiple attempts to stand up but fails completely to achieve the desired functionality as seen in the top subfigure. The error plot clearly shows that the error magnitude crosses the detection threshold of 0.6 indicating presence of errors in the system.

Figure 4.29 illustrates a fault injected balancing experiment with a sensor fault of 5%. This destabilizes the balancing algorithm resulting in oscillations of the robot around the equilibrium point as seen in the top subfigure. The bottom subfigure demonstrates that the error signal is significantly higher than the chosen threshold of $e_{th} = 0.6$ indicating prompt detection of a system fault without complete failure. This is particularly helpful for detection of early degradation of sensors and actuators before catastrophic failure occurs. The experimental data from a real autonomous system proves the effectiveness of the proposed methodology.

4.6 Summary

The presented simulation results and hardware data demonstrate the effectiveness and practical feasibility of the proposed methodology for error detection in nonlinear state space control systems operating under arbitrary failure mechanisms. Computational overhead and error coverage data from two test cases conclusively prove the benefits of the presented scheme.

CHAPTER 5

ALERA: ACCELERATED LEARNING ENABLED REINFORCEMENT ARCHITECTURE FOR ERROR CORRECTION IN NONLINEAR CONTROL SYSTEMS

Adaptive dynamic programming (ADP), proposed by Werbos [129] approximates the performance index function forward-in-time and generates an intelligent method of approximate optimal control for nonlinear systems [130, 131]. Reinforcement learning (RL) [132] is such a framework in which a controller optimizes its behavior by interacting with its environment. After taking an action, known as *policy*, the controller receives a scalar reward from the environment, that provides an evaluation of the quality of that action. The user sets a certain goal by specifying a suitable reward function for the RL controller, and the RL controller then learns to maximize the cumulative reward received over time, known as the *value function* in order to reach that goal. A typical RL controller starts learning without any knowledge, initially reacting to external stimulus in non-optimal ways through trial-and-error, while learning through such experiences to develop an optimal control policy. This leads to a long period of unpredictable and potentially damaging behavior. The learning rate needs to be considerably accelerated for RL controllers to be useful in practical nonlinear control systems.

Actor-critic methods are a class of RL algorithms that learn a critic function as the value function approximator and a separate actor function as the policy approximator. Such actor-critic methods have recently been used as optimal controllers in a variety of nonlinear control problems [133–136]. However, no previous research has focused on using actor-critic networks as self-learning controllers in the presence of parametric failures and malfunction in the system. Prior works have emphasized acceleration of the learning rate for a nominal system [137, 138]. In this chapter, the error detection methodology described in the

previous chapter is exploited to enable the usage of RL controllers as augmentation to a nonlinear controller and significantly speed up the self-learning capabilities of actor-critic methods for quick recovery of system performance in real-time.

5.1 Preliminaries

This section discusses the framework of RL for deterministic Markov decision process (MDP) and introduces the actor-critic methods as approximate RL algorithms.

5.1.1 Reinforcement Learning

In general, a RL problem is described as a stochastic MDP modeled by the tuple $M(X, U, f, \rho)$ where X is the state space, U is the action space, $f : X \times U \times X \mapsto [0, \infty)$ is the transition probability function and $\rho : X \times U \times X \mapsto \Re$ is the reward function. In this work, the RL is used in a deterministic framework and hence we redefine the MDP tuple as $M(X, U, \bar{f}, \bar{\rho})$ where X is the state space of the control system, U is the space of control inputs, $\bar{f} : X \times U \mapsto X$ is the state transition function that describes the system dynamics and $\bar{\rho} : X \times U \mapsto \Re$ provides the reward function, assumed to be bounded. At each time step t, the controller takes an action $\mathbf{u}(t) \in U$ depending on the system state $\mathbf{x}(t) \in X$ from a separate control policy $\pi : X \mapsto U$. The next system state $\mathbf{x}(t+1)$ is determined by the state transition function $\bar{f}(\mathbf{x}(t), \mathbf{u}(t))$. After the transition to $\mathbf{x}(t+1)$, the controller receives a scalar reward $r(t+1)$ according to the reward function $\bar{\rho} : r(t+1) = \bar{\rho}(\mathbf{x}(t+1), \mathbf{u}(t+1))$. The objective of a RL algorithm is to find an optimal policy that maximizes the cumulative future rewards over an infinite horizon of time. The discounted sum of such rewards is known as *return* and defined as the continuous value function $V^{\pi} : X \mapsto \Re$ as

$$V^{\pi}(\mathbf{x}(t)) = \int_{t}^{\infty} e^{-\frac{s-t}{\tau}} \bar{\rho}(\mathbf{x}(s), \mathbf{u}(s)) ds \tag{5.1}$$

for any initial state $\mathbf{x}(t)$ and τ is the time constant for discounting future rewards. The fading exponential function puts reducing emphasis on future rewards by weighing present rewards at higher value. The value function at any state provides a quantitative measure of how 'good' the present state is for maximizing the future return. According to the principle of optimality, the condition for the optimal value function at time t is given as

$$\frac{1}{\tau}V^*(\mathbf{x}(t)) = \max_{\mathbf{u}(t)\in U}\left[\bar{\rho}(\mathbf{x}(t),\mathbf{u}(t)) + \frac{\partial V^*(\mathbf{x})}{\partial \mathbf{x}}\bar{f}(\mathbf{x}(t),\mathbf{u}(t))\right] \tag{5.2}$$

where (5.2) denotes the Hamilton-Jacobi-Bellman (HJB) equation for infinite-horizon, discounted reward problems (derivation shown in appendix A and originally explored in [139]). The optimal policy $\pi^*(\mathbf{x}(t))$ in a RL problem is determined by the action that maximizes the right-hand side of (5.2) as

$$\pi^*(\mathbf{x}) = \arg\max_{\mathbf{u}\in U}\left[\bar{\rho}(\mathbf{x},\mathbf{u}) + \frac{\partial V^*(\mathbf{x})}{\partial \mathbf{x}}\bar{f}(\mathbf{x},\mathbf{u})\right] \tag{5.3}$$

Applying an action according to the policy determined in (5.3) ensures that the return for the particular choice of reward function $\bar{\rho}$ is maximum along the entire trajectory $\mathbf{x}(t) \quad \forall t$. Thus, the primary goals of a RL algorithm are: i) estimate the value function $V(\mathbf{x})$ based on the received rewards after taking actions drawn from the policy $\pi(\mathbf{x})$ and ii) updating the policy $\pi(\mathbf{x})$ according to the current estimate of the value function $V(\mathbf{x})$. Actor-critic methods are popular schemes for implementing a RL framework and chosen as our methods in this work.

5.1.2 Actor-Critic Methods

Actor-critic algorithms [140] contain two distinct learning units - a *critic* that tries to learn the value function of the policy and an *actor* that provides the action according to a certain policy. Popular critic-only methods such as Q-learning [141–143] and SARSA [144, 145]

implement a state-action value function $Q(\mathbf{x}, \mathbf{u})$ as a critic and no explicit actor policy. The value function Q evaluates the quality of choosing a particular action in a particular state with respect to the net return. The optimal deterministic policy is computed by performing an optimization procedure on the learned value function. Such methods do not ensure the near-optimality of the resulting policy when learning in an online setting that weakens the prospect of using such techniques for real-time control. On the contrary, actor-only methods such as the stochastic real-valued (SRV) [146] and REINFORCE [147] algorithms do not explicitly estimate a value function and use policy gradient methods for computing the best action. Such parametrized policies are updated by performing optimization of the return defined in (5.1) directly over the parameter space of the policy. The two major advantages of actor-only methods are that the parametric optimization allows generation of actions in the complete continuous action space, thus exploring a wide range of action choices in a control system and the gradient descent methods enforce strong convergence with appropriate choices of learning rates [145, 148]. The major drawback of these approaches are that every gradient is computed without utilizing knowledge of prior estimates [149, 150] that elongates the learning process due to repetitive exploration of the action space and makes these methods unsuitable for implementation in control systems.

Actor-critic techniques introduced in [151] and henceforth investigated in [149, 152, 153] combine the advantages of actor-only and critic-only methods, thus making them attractive candidates for implementation as controllers. These methods are capable of generating continuous actions like actor-only methods, while the policy gradients are assisted by the evaluation of the critic enabling fast convergence. The critic evaluates the current actor policy and updates the value function estimate using data samples. The current value function estimate is then utilized for improving the policy parameters in the direction of performance improvement. A schematic structure of an actor-critic algorithm is illustrated in Figure 5.1. The actor generates the policy action $\mathbf{u}$ depending on the current state $\mathbf{x}$. The critic receives the reward r for applying the current action and evaluates the quality of the

present policy by updating the value function estimate. After a few policy evaluation steps by the critic, the actor is updated using gradient descent on the parameter space.

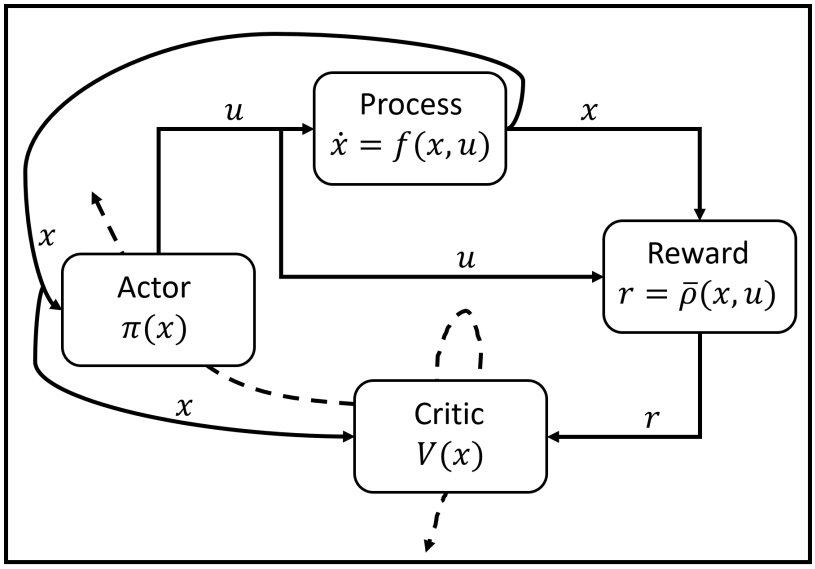


Figure 5.1: Schematic overview of an actor-critic algorithm. The dashed line indicates the updating of critic and actor by the critic output.

The actor-critic methods utilize the popular framework of temporal difference (TD) learning [145]. Differentiation of (5.1) by t creates a self-consistency condition as,

$$\dot{V}^{\pi}(\mathbf{x}(t)) = \frac{1}{\tau} V^{\pi}(\mathbf{x}(t)) - r(t) \tag{5.4}$$

that is valid for any policy (including the optimal policy of (5.3)). Any imperfect estimate of the value function violates the consistency condition in (5.4) generating the TD error,

$$\delta(t) \equiv r(t) - \frac{1}{\tau} V(t) + \dot{V}(t) \tag{5.5}$$

At each step of the learning algorithm, the value function estimate is adjusted to reduce the TD error $\delta(t)$. Thus the magnitude of the TD error $\delta(t)$ provides a quantitative estimate of how different the current value function estimate is from the actual value function. The convergence of the value function and the policy to their optimal values is signified by the TD error approaching zero. Hence the TD error is exploited for gradient descent update

[145] of the critic parameters. The learning is further accelerated by the use of *eligibility traces* that introduce a notion of memory in the learning procedure. To define eligibility traces, the TD error in (5.5) is expressed in its discretized form as

$$\delta_t = r_t + \gamma V_t - V_{t-1} \tag{5.6}$$

where $\gamma = 1 - \frac{\Delta t}{\tau} \simeq e^{-\frac{\Delta t}{\tau}}$ is the discretized discount factor with Δt as the sampling duration and the variable values are scaled as $V_t = \frac{1}{\Delta t} V(t)$. Eligibility trace is an additional memory variable $z_t(\mathbf{x})$ associated with each state $\mathbf{x}$ that evolves with time t. At each learning step, the eligibility traces for all states decay by $\gamma\lambda$ and the eligibility trace for the current state visited is incremented by 1 as

$$z_t(\mathbf{x}) = \begin{cases} \gamma\lambda z_{t-1}(\mathbf{x}), & \text{if } \mathbf{x} \neq \mathbf{x}_t \\ \gamma\lambda z_{t-1}(\mathbf{x}) + 1, & \text{if } \mathbf{x} = \mathbf{x}_t \end{cases} \tag{5.7}$$

where $\lambda \in [0, 1]$ is the trace decay parameter. At any time, the eligibility traces record the past history of recently visited states and indicate the degree to which each state is eligible for undergoing learning changes. This evaluates the trajectory of system states instead of each particular state in response to the current policy. This is due to the fact that the reward r_t is received as a result of a series of actions taken. Hence, eligibility traces are used to assign credit to states visited several steps earlier and provides multiple-step backup for distributing the reinforcement learning.

5.1.3 Function Approximators as Actor-Critic Units

The value function and the policy definitions in (5.1) and (5.3) assume exact continuous representations over the state space that are not feasible in practical systems due to the infinite size of continuous state or action spaces [154]. In practical implementations of RL

algorithms, function approximators [145] such as linear coding, tile coding, radial basis functions, Kanerva coding, etc. are used to represent an actor and a critic due to the infinite size of continuous state or action spaces. In any function approximator, the value function of the critic and the actor policy are parametrized by the respective parameter vectors $\theta^c \in \Re^{n_c}$ and $\theta^a \in \Re^{n_a}$ as $V_{\theta^c}(\mathbf{x})$ and $\pi_{\theta^a}(\mathbf{x})$. The TD error δ_t defined in (5.6) is utilized for performing a gradient descent update of the critic parameter as,

$$\theta^c_{t+1} = \theta^c_t + \eta^c \delta_t \nabla_{\theta^c} V_{\theta^c_t}(\mathbf{x}_t) \tag{5.8}$$

where $\eta^c > 0$ is the learning rate of the critic. Use of eligibility traces defined in (5.7) update the critic in a TD(λ) algorithm [145] as

$$\theta^c_{t+1} = \theta^c_t + \eta^c \delta_t z_t \tag{5.9}$$

where the eligibility trace is defined as a vector $z_t \in \Re^{n_c}$ instead of storing individual trace history for each state. The eligibility trace update equation [149] is

$$z_t = \gamma \lambda z_{t-1} + \nabla_{\theta^c} V_{\theta^c_t}(\mathbf{x}_t) \tag{5.10}$$

Similarly, the actor parameter update equation is

$$\theta^a_{t+1} = \theta^a_t + \eta^a \delta_t \nabla_{\theta^a} \pi_{\theta^a_t}(\mathbf{x}_t) \tag{5.11}$$

where $\eta^a > 0$ is the learning rate of the actor. Though eligibility traces are popular in actor-only methods, they are rarely used for policy update in actor-critic methods. Equations (5.9-5.11) define the basic framework of RL for parametric function approximators.

5.1.4 Normalized Gaussian network based AC methods

In this work, normalized Gaussian networks are used as function approximators [139] for implementation of critic and actor. The value function is represented by the parameter vector $\mathbf{w}^c$ as

$$V(\mathbf{x}, \mathbf{w}^c) = \sum_{k=1}^{K} w_k^c b_k(\mathbf{x}) \tag{5.12}$$

where

$$b_k(\mathbf{x}) = \frac{a_k(\mathbf{x})}{\sum_{l=1}^{K} a_l(\mathbf{x})}, \quad a_k(\mathbf{x}) = e^{-||\mathbf{s}_k^{\intercal}(\mathbf{x}-\mathbf{c}_k)||^2}$$

The vectors $\mathbf{c}_k$ and $\mathbf{s}_k$ define the center and the size of the kth Gaussian basis function. The number of basis functions K denote the level of discretization of the continuous state space and is a design choice. The TD error δ_t and eligibility trace $\mathbf{z}_t$ are defined in (5.6) and (5.10). The update equation of the critic parameter is then similar to (5.9 as

$$\mathbf{w}_{t+1}^c = \mathbf{w}_t^c + \eta^c \delta_t \mathbf{z}_t \tag{5.13}$$

The actor policy is also implemented as a normalized Gaussian network based approximator $\pi(\mathbf{x}, \mathbf{w}^a)$ parametrized by $\mathbf{w}^a$ and given as

$$\begin{aligned} \mathbf{u}_t &= \mathbf{u}^{\max} s\Big(\pi(\mathbf{x}_t, \mathbf{w}_t^a) + \mathbf{n}_t\Big) \\ &= \mathbf{u}^{\max} s\Bigg(\sum_k w_{k(t)}^a b_k(\mathbf{x}_t) + \mathbf{n}_t\Bigg) \end{aligned} \tag{5.14}$$

where $\mathbf{u}^{\max}$ is the maximum action possible, $\mathbf{n}(t)$ is a stochastic term that is initially used for exploration of the parametric space for evaluating new policies and gradually reduced to zero with progress of learning and $s(.)$ is a component-wise sigmoid function. The update

equation of the actor parameter is similar to (5.11) as

$$\mathbf{w}_{t+1}^{a} = \mathbf{w}_{t}^{a} + \eta^{a}\delta_{t}\nabla_{\mathbf{w}^{a}}\pi_{\mathbf{w}_{t}^{a}}(\mathbf{x}_{t}) \tag{5.15}$$

The above equations demonstrate that an actor-critic network is completely defined by its weight vectors $\mathbf{w}^{c}$ and $\mathbf{w}^{a}$. Similar to other gradient descent algorithms, actor-critic learning depends on the exploration term $\mathbf{n}(t)$ to evade out of local optima. The proposed research in this work intends to reduce this exploration significantly by initializing the actor-critic weight vectors to appropriate parameters, thus accelerating the self-learning process and enabling fast system adaptation under faults or unforeseen situations. The full implementation of the AC algorithm is shown in Algorithm 2.

Algorithm 2 Gaussian Network based AC Methods

procedure AC ALGORITHM
  Input: $\gamma, \lambda, \eta^{c}, \eta^{a}, n_{c}, n_{a}$
  $K \leftarrow$ Number of basis functions
  $s(.) \leftarrow$ Sigmoid function definition
  for $k = 1$ to K **do**
    $\mathbf{s}_{k} \leftarrow$ Size of kth Gaussian basis function
    $\mathbf{c}_{k} \leftarrow$ Center of kth Gaussian basis function
    $a_{k}(\mathbf{x}) = e^{-||\mathbf{s}_{k}^{\intercal}(\mathbf{x}-\mathbf{c}_{k})||^{2}}$
    $b_{k}(\mathbf{x}) = \frac{a_{k}(\mathbf{x})}{\sum_{l=1}^{K} a_{l}(\mathbf{x})}$
  Initialize $\mathbf{x}_{0}, \mathbf{w}_{0}^{c}, \mathbf{w}_{0}^{a}$
  $\mathbf{z}_{0} = \mathbf{0}$
  Apply random input $\mathbf{u}_{0}$
  $i \leftarrow 1$
  while Objective not met **do**
    Measure $\mathbf{x}_{i}, r_{i}$
    $\mathbf{u}_{i} \leftarrow \mathbf{u}^{\max} s\left(\sum_{k} w_{k(i-1)}^{a} b_{k}(\mathbf{x}_{i}) + \mathbf{n}_{i} \right)$
    Apply $\mathbf{u}_{i}$
    $\delta_{i} \leftarrow r_{i} + \gamma \sum_{k} w_{k(i-1)}^{c} b_{k}(\mathbf{x}_{i}) - \sum_{k} w_{k(i-1)}^{c} b_{k}(\mathbf{x}_{i-1})$
    $\mathbf{z}_{i} \leftarrow \gamma\lambda\mathbf{z}_{i-1} + \nabla_{\mathbf{w}^{c}} V(\mathbf{x}_{i}, \mathbf{w}_{i-1}^{c})$
    $\mathbf{w}_{i}^{c} \leftarrow \mathbf{w}_{i-1}^{c} + \eta^{c}\delta_{i}\mathbf{z}_{i}$
    $\mathbf{w}_{i}^{a} \leftarrow \mathbf{w}_{i-1}^{a} + \eta^{a}\delta_{i}\nabla_{\mathbf{w}^{a}}\pi(\mathbf{x}_{i}, \mathbf{w}_{i-1}^{a})$
    $i \leftarrow i + 1$

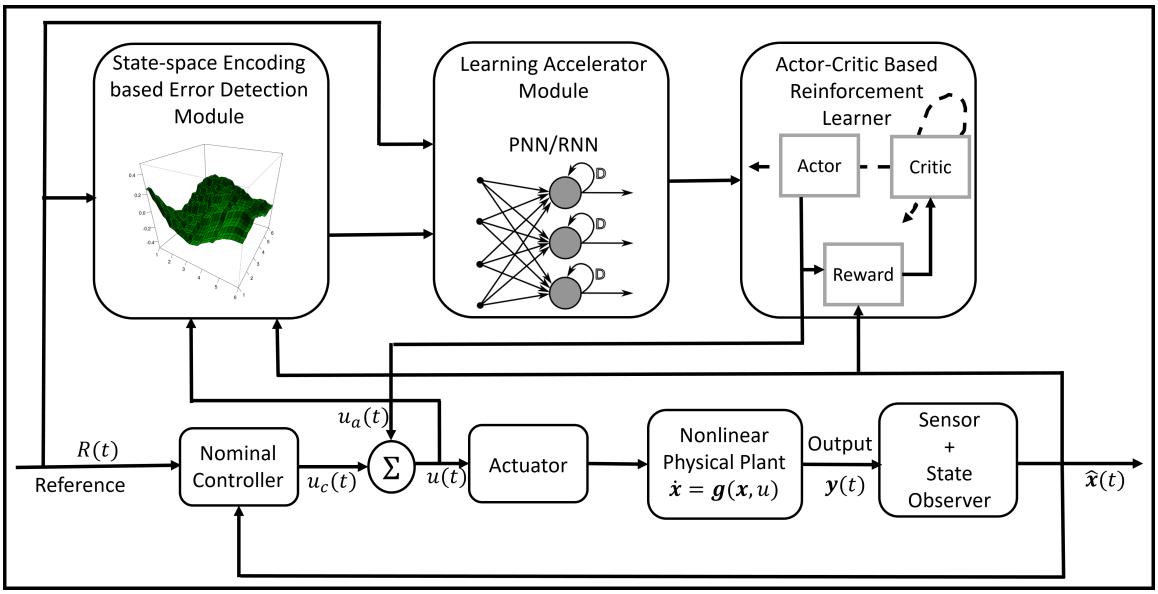


Figure 5.2: ALERA: Accelerated Learning Enabled Reinforcement Architecture

5.2 ALERA: Accelerated Learning Enabled Reinforcement Architecture

The proposed architecture is illustrated in Figure 5.2. ALERA is used as an augmentation to the nominal controller for fast reconfiguration of effective control law in erroneous situations. The nonlinear physical plant with system dynamics of $\dot{\mathbf{x}} = \mathbf{g}(\mathbf{x}, u)$ represents the actual process to be controlled. The sensor measures the output $\mathbf{y}(t)$ and the state observer (nonlinear observer or extended Kalman filter [155]) estimates the system states $\mathbf{x}(t)$ from the measurement $\mathbf{y}(t)$. The predicted state $\hat{\mathbf{x}}(t)$ is used along with the externally provided reference input $R(t)$ in the nominal controller $u_c(t) = h(\hat{\mathbf{x}}(t), R(t))$ to generate the required input $u_c(t)$ that is applied to the plant through the actuators. The control law $h(.)$ is designed to meet certain specifications according to the design requirements. ALERA augments the nominal controller by providing an additional control input $u_a(t)$, thus applying the composite action $u(t) = u_c(t) + u_a(t)$. The nominal controller is designed to meet the system specifications under the assumption that all physical components of the system perform accurately. Under fault mechanisms, the nominal controller is sub-optimal and cannot meet the system specifications and may even fail to achieve system functionality

under critical errors. ALERA is designed to compensate for such effects in real-time with low latency. The primary modules of ALERA are: i) error detection module, ii) learning accelerator module, and iii) actor-critic based reinforcement learner.

5.2.1 Error Detection Module

The error detection is accomplished by the state-encoding based methodology described in the previous chapter. It has been demonstrated in linear systems [156] that the transient waveform of the error signal contains diagnostic information and critical system parameters can be predicted with reasonable high accuracy. This diagnostic capability of the error signal $e(t)$ is exploited in the learning accelerator module for fast reconfiguration of control law in real-time. Before describing the learning accelerator module, the actor-critic reinforcement learner is discussed next.

5.2.2 Actor-Critic based Reinforcement Learner

The actor-critic reinforcement learner is an augmentation to the nominal controller as shown in Figure 5.2. The nominal controller with output $u_c(t)$is designed for the physical plant to satisfy certain system specifications. The estimated state $\hat{\mathbf{x}}(t)$ along with the reference input $R(t)$ is used to compute the reference tracking error $e_R(t)$ and the required control input $u_c(t)$ to reduce the tracking error. In a nominal fault-free system, the reinforcement learner is initially trained online with the nominal controller present in the control loop for improved closed-loop tracking performance [157]. The reward function for the reinforcement learner is derived from the estimated state $\hat{\mathbf{x}}(t)$ and the actor policy $u_a(t)$ such that the actor-critic algorithm assumes the nominal controller as part of the environment with which it reacts and learns the actor-critic units appropriately. In a system compromised by faults or in anomalous situations, the TD error $\delta(t)$ of the trained nominal reinforcement learner becomes non-zero indicating exertion of sub-optimal control policy. The learning algorithm of the actor-critic network is triggered by the non-zero TD error

to adapt to a different actor policy by reconfiguring the actor and critic weight vectors $\mathbf{w}^a$ and $\mathbf{w}^c$ such that the return defined by the reward function is maximized in the altered environment. However, due to the exploratory nature of the actor policy indicated by the stochastic term $\mathbf{n}_t$ in (5.14) and the episodic task based updates inherent in reinforcement learning, the optimal policy can take significant time to converge. The learning accelerator module is designed and trained to reinitialize the actor-critic weights to reduce the latency of learning. The schematic of the actor-critic reinforcement learner module is shown in Figure 5.3.

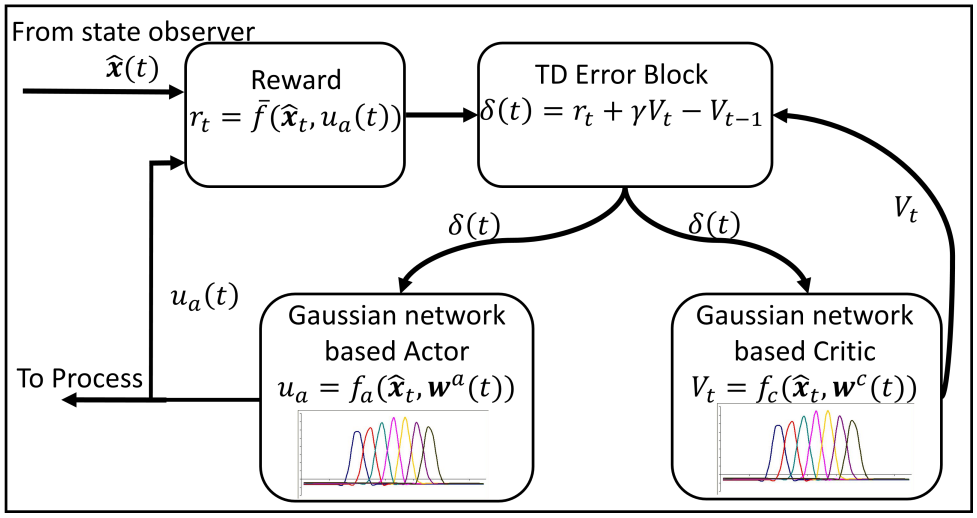


Figure 5.3: Schematic of the actor-critic based reinforcement learner module

5.2.3 Learning Accelerator Module

The schematic of the learning accelerator module is shown in Figure 5.4. The primary objective of the accelerator module is to predict actor and critic parameters from the error signal $e(t)$ along with the reference input $R(t)$ (since the error signal $e(t)$ depends on the input stimulus to the system). This learning acceleration is accomplished by a pre-trained probabilistic neural network (PNN) or a recurrent neural network (RNN) depending on the pre-deployment simulation capabilities and training effort that can be afforded. The tradeoffs between the two schemes are discussed later.

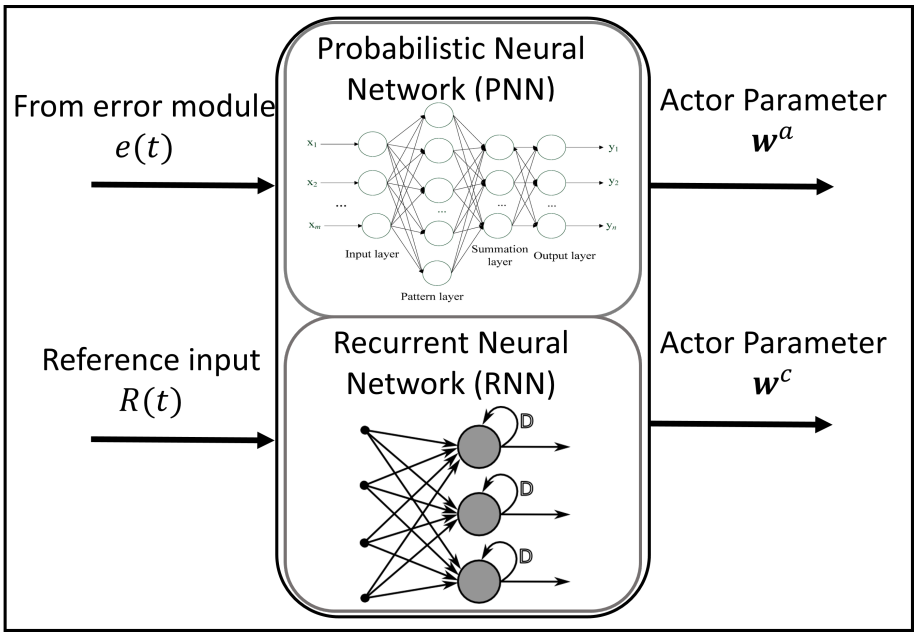


Figure 5.4: Schematic of the learning accelerator module. It consists of either a PNN or a RNN depending on the pre-deployment training cost required

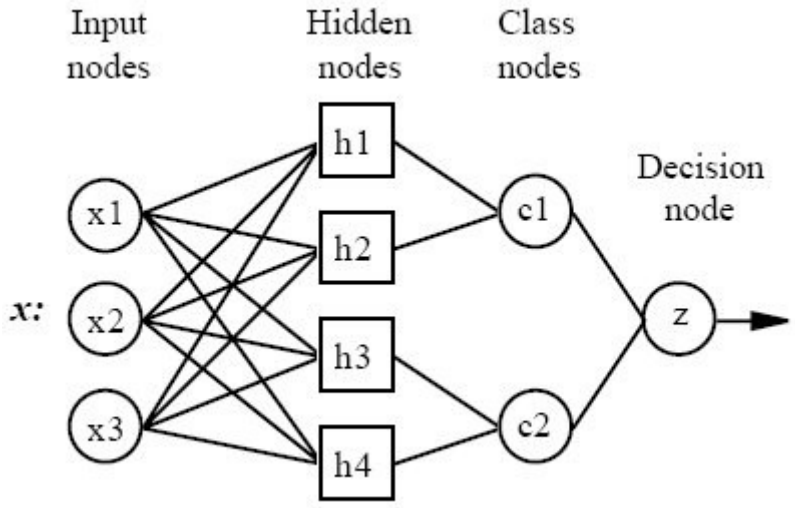


Figure 5.5: Schematic diagram of a PNN

i) PNN - The schematic of a PNN architecture is shown in Figure 5.5. PNNs are functionally similar to k-nearest neighbor (k-NN) models [insert citation] where the basic idea is to assign a class to an input item depending on the learned classification of training vectors. PNNs are used in this research because it is much faster to train a PNN than a typical multilayer perceptron network and PNNs are relatively insensitive to outliers [insert cita-

tion]. The input layer contains N neurons where N is the size of the input vector. After standardizing the range of input values, the input neurons feed the values to the hidden layer neurons. The number of neurons in the hidden layer is equal to the number of cases in the training data set. Each of the neurons stores the input data from the training set along with the target class. Presented with a new test vector from the input layer, each hidden neuron computes the ℓ_2-norm of the test vector from the neuron's data and applies a radial basis function (RBF) to the distance to compute the weight of each training point with respect to the test data. The next layer consists of one pattern neuron for each target class. The weighted value from the hidden neuron is fed to the pattern neuron that corresponds to the neuron's category. All the class nodes add the values for the category they represent and finally the decision layer compares the weighted votes for each target class accumulated in the pattern layer and uses the largest vote to predict the target class for the input data.

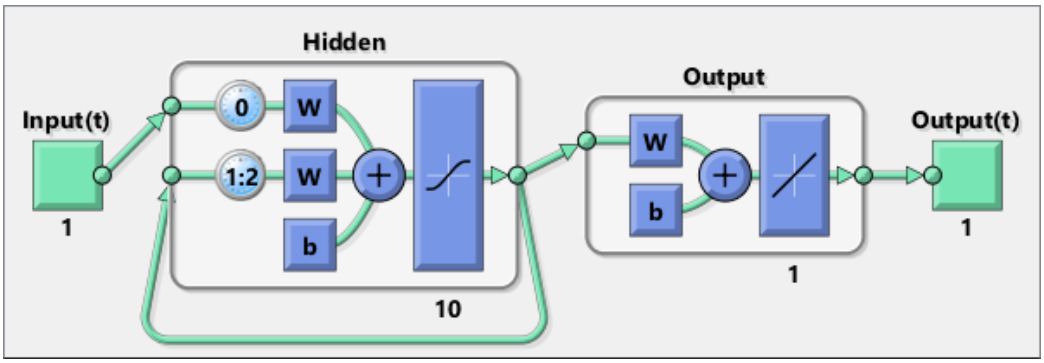


Figure 5.6: Schematic diagram of a RNN

ii) RNN - The schematic of an RNN architecture is shown in Figure 5.6. The primary idea behind RNNs are to exploit sequential information while mapping any functionality. Unlike traditional feedforward neural networks, RNNs implement a recurrent connection with a tap delay associated with it. This allows the network to have an infinite dynamic response to time series input data and integrate 'memory' in the learning procedure. The schematic of an RNN shown in Figure 5.6 shows that it is structurally similar to a feedforward neural network with an additional feedback of the hidden layer output to the input of the hidden layer neurons. This enables the learning of nonlinear mapping from a temporal

data sequence to a target vector associated with each sequence.

With this description of the different components of ALERA, we next focus on the proposed methodology of real-time accelerated self-learning for systems operating in anomalous situations or under arbitrary failure modes.

5.3 Proposed Methodology: Real-time Self-learning

The key idea of the real-time accelerated self-learning is to store pre-learned critic and weight vectors for a variety of faulty situations and reinitialize the AC unit with the stored data based on the error signal $e(t)$, thus assisting the learning procedure and applying compensating measures. The proposed methodology of real-time self-learning is categorized into i) pre-deployment scheme where training of the regression methods in the error detection module, neural networks in the accelerator block and the actor-critic unit in the reinforcement learner are performed and ii) post-deployment scheme where real-time fast reconfiguration of applied control is adopted.

5.3.1 Pre-deployment

The primary objective of pre-deployment simulations is to train the components of ALERA to tackle scenarios with assumed failure mechanisms such that it adapts to unforeseen situations with low latency. The nominal controller is designed to meet system specifications of a fault-free system and the reinforcement learner is trained with different inputs $R(t)$ for achieving tighter reference tracking and ensuring better closed-loop system stability. During this phase, the AC unit performs episodic experiments on the system and both the critic and the actor learn the optimal value function V^* and policy π^* with convergence to nominal weight vectors $\mathbf{w}^c_{(nom)}$ and $\mathbf{w}^a_{(nom)}$ respectively. The completion of learning is indicated by the TD error $\delta(t)$ converging to zero as discussed in Section 5.1.2. The MARS regression functions in the error detection module are trained with a set of expected reference inputs such that the fault-free error signal $e(t)$ is minimal and lies below a cer-

tain threshold. The training of the error detection module is discussed in details in [TDSC paper]. Based on the assumed failure models and parametric degradations of the system (plant, sensors, actuators and mixed-signal interfaces), a finite fault universe is created. A Monte Carlo sampling of the system is performed over the parameter space of the assumed fault models, thus generating a set of fault-injected systems. The sampling distribution and granularity depends on the computational effort required to comprehensively simulate each fault-injected system. For each of these systems, three features are observed - i) fault model parameter set, ii) transient error signal along with the system input $R(t)$ for a particular time duration from the instant of error detection, and iii) TD error $\delta(t)$ for the RL unit during the same duration. Computational overhead and latency of correction procedure are traded off with diagnosability in determining the observation window length. Observation of the transient error signal over an elongated duration is helpful in diagnosing the probable cause of error with higher accuracy but delays the compensation procedure along with placing higher computational burden on the accelerator module. A regression model is built with the error and input signal samples as inputs and the fault parameter set as output such that critical parameters can be predicted from the transient error signal. This regression model is required only for the PNN-based scheme and is a part of the learning accelerator module whereas the RNN-based scheme directly operates on the time-series data of the error signal and the reference input.

The nominal AC unit for a fault-injected system generates non-zero TD signal $\delta(t)$ indicating a sub-optimal policy and starts the self-learning procedure to adapt to new critic and actor parameter vectors by employing the gradient learning steps of (5.13) and (5.15). The non-zero TD error trajectory indicates the degree of sub-optimality and signifies how far the nominal weight vectors $\mathbf{w}^c_{(nom)}$ and $\mathbf{w}^a_{(nom)}$ are from the new optimal values for the altered system. Hence the Monte Carlo samples are clustered into L classes in the fault parameter set based on the summed TD error magnitude during the learning of the nominal AC network over the chosen time duration. The PNN is trained with the class

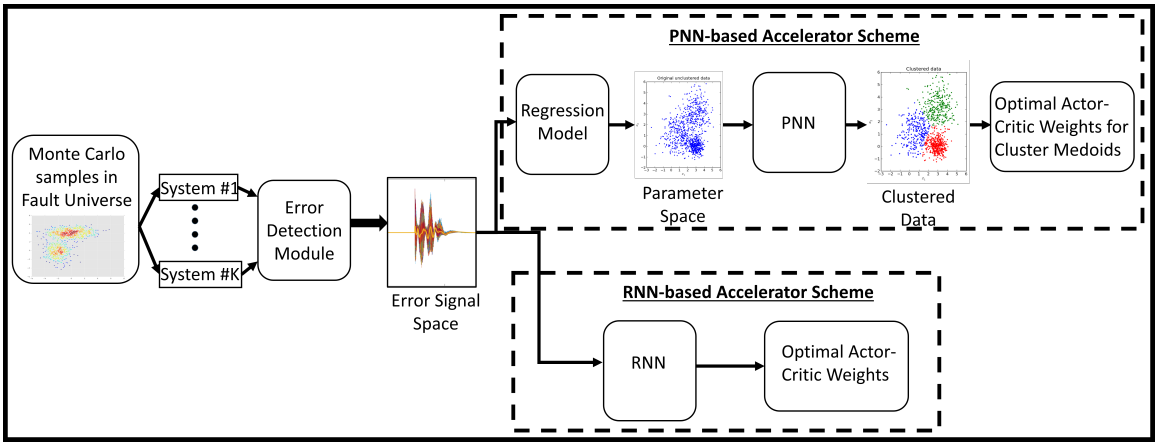


Figure 5.7: Schematic work-flow of the pre-deployment training procedure for both the PNN and the RNN-based schemes

labels for each Monte Carlo sample. For each of the L clusters, a cluster medoid is selected to represent one particular fault injected system that best reflects the learning performance of the nominal AC unit for all the systems in that cluster. Unlike centroids, medoids are always members of the data set and does not represent a sample that may not be a feasible system. Next, for each of these L cluster medoids, an optimal AC network is learned such that the TD error $\delta(t)$ converges to zero, thus satisfying the design specifications in presence of injected faults. Each of the cluster medoids generates a different set of actor and critic parameter vectors $\mathbf{w}_l^a$ and $\mathbf{w}_l^c \; \forall \, l = 1$ to L. The number of clusters L is an important metric in deciding the benefits achievable through the learning acceleration. With an increase in L, the number of fault injected systems for which an optimal AC network is learned increases. However, the computational effort in training a new AC for each of the cluster medoids also increases along with the memory requirement for storing the AC parameters for each of those learned units. As the cluster count increases beyond a certain point, the achievable latency benefits of real-time compensation by reinitializing the Ac unit from pre-stored individual learned AC weights for each cluster center are not sustainable due to the high memory requirements. The RNN-based accelerator scheme is proposed to represent the other end of the spectrum of increasing cluster counts. In the RNN-based scheme, M fault-injected systems are created by carefully sampling the fault universe. For each of these M

systems, an optimal AC unit is trained for best control performance generating M different actor and critic weights $\mathbf{w}_i^a$ and $\mathbf{w}_i^c$ $\forall$ $i = 1$ to M. The RNN is trained with the time series data of the error signal $e(t)$ along with the input $R(t)$ for each of the M systems as input data and the respective actor and critic weights as output data. This enables the RNN to learn a nonlinear mapping from the temporal signal waveform of $e(t)$ and $R(t)$ such that actor and critic weights can directly be predicted from an arbitrary error signal data of a fault-injected system without the need for any intermediate parametric diagnosis and regression model. The schematic work-flow of the pre-deployment training is shown in Figure 5.7.

5.3.2 Post-deployment

In the post-deployment operational phase, the nominally trained AC unit serves as an augmentation to the nominal controller. In the presence of faults or anomalies, the MPCS based error detection module generates an error signal $e(t)$ at $t = t_0$ above the predetermined threshold, indicating that an error or an unlearned event has occurred. In the PNN-based accelerator scheme, the pre-trained regression model shown in Figure 5.7 predicts the critical system parameters from the error signal and system input recorded over a short time duration until $t = t_1$. During the time duration $t_0 \rightarrow t_1$, the nominal AC unit has started the reinforcement learning tasks to adapt to the new situation. The PNN categorizes the system behavior to one of the clusters in the parameter space and the AC network is initialized with the weight vectors corresponding to that cluster medoid at time $t = t_2$. In an RNN-based scheme, the RNN is invoked at $t = t_1$ directly with the error $e(t)$ and input signal as time series data input to compute the actor-critic weights directly. If the TD error $\delta(t)$ increases in magnitude after this reinitialization, it indicates that the AC unit has better adapted from the learning experience between t_0 and t_2 and the reinitialization procedure deteriorates the learning process. In such a situation, the reinitialization is not adopted and the usual reinforcement learning is continued. However, if the TD error $\delta(t)$ magnitude

reduces due to reinitialization, it indicates an accelerated boost to the learning process and significantly reduces the exploratory steps needed by the reinforcement learning algorithm for an affected system. The steps of error detection and control compensation are shown in Figure 5.8.

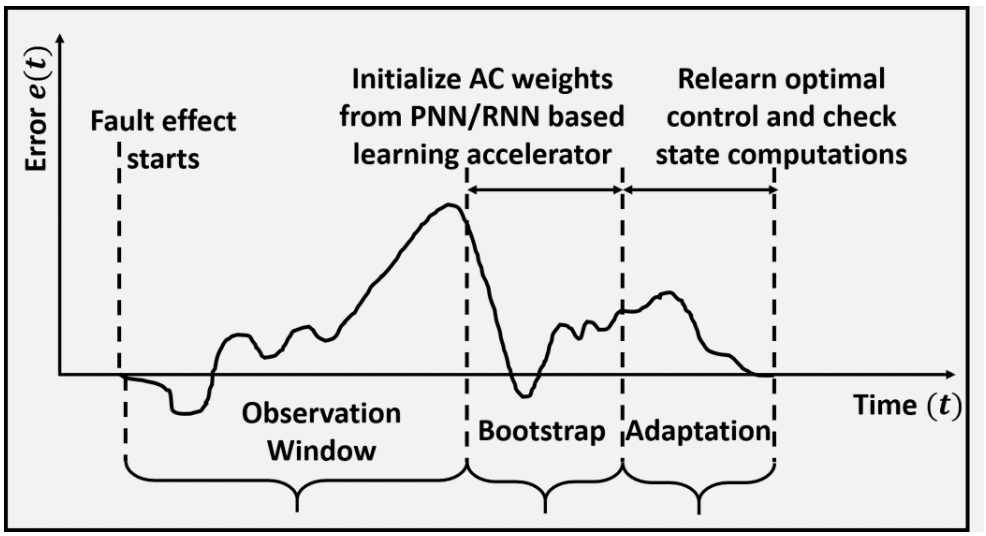


Figure 5.8: Detection, Diagnosis and Repair Cycle

5.4 Fault Models

The fault models in this work consider malfunctions in sensors and actuators in addition with altered physical environment model signifying operational anomaly. The sensor and actuator faults are modeled [158] as parametric degradations. The nominal controller relies its control computation on the tracking error derived from the sensor measurement and applies the required action to the plant through actuator mechanism. Any corruption in the sensor or actuator functionality degrades the system behavior due to the incapability of the controller to exert appropriate control on the system. Similarly, the control policy is nominally designed under the assumption of system operating in a set of plausible environmental conditions as all feasible operating conditions cannot be foreseen. Under a different environmental condition, the nominal control performance is not adequate to achieve complete system functionality with the desired specifications. The fault models are shown in

Figure and described as

i) Parametric degradations - Sensor and actuator degradations are generally not abrupt and the performance wear-off is gradual. The mathematical model of temporal degradation of a parameter p is expressed as:

$$\tilde{p}(t)|_{t>\tau_p} = p(t)|_{t<\tau_p} \times \alpha e^{\beta(t-\tau_p)} \tag{5.16}$$

where $t = \tau_p$ is the start of parametric degradation, $p(t)|_{t<\tau_p}$ is the nominal value of parameter p before $t = \tau_p$, $\tilde{p}(t)|_{t>\tau_p}$ is the degraded parameter value and α and β denote the degree and rate of degradation. This parametric perturbation model alters the sensor/actuator parameters $p_1, p_2, \cdots, p_n$ into a modified set $\tilde{p_1}, \tilde{p_2}, \cdots, \tilde{p_n}$ such that the corresponding transfer function of the entire system is changed in a way that is not well modeled by additive disturbances alone.

ii) Operating environment change - With change of operating environment, the system dynamics shown in Figure 5.2 is altered as,

$$\dot{\mathbf{x}}(t) = \mathbf{g}'(\mathbf{x}(t), u(t)) \tag{5.17}$$

where $\mathbf{g}'$ is the modified system dynamics due to a change in the operating environment. The nominal controller $h(.)$ and the AC unit are designed and trained for the system dynamics $\mathbf{g}(.)$ and fail to provide optimal control for the altered dynamics $\mathbf{g}'(.)$. Instead of arbitrarily modifying the function $\mathbf{g}(.)$, a different state evolution dynamics is computed to model the operational anomaly.

5.5 Test Case I: Inverted Pendulum System

5.5.1 System Description

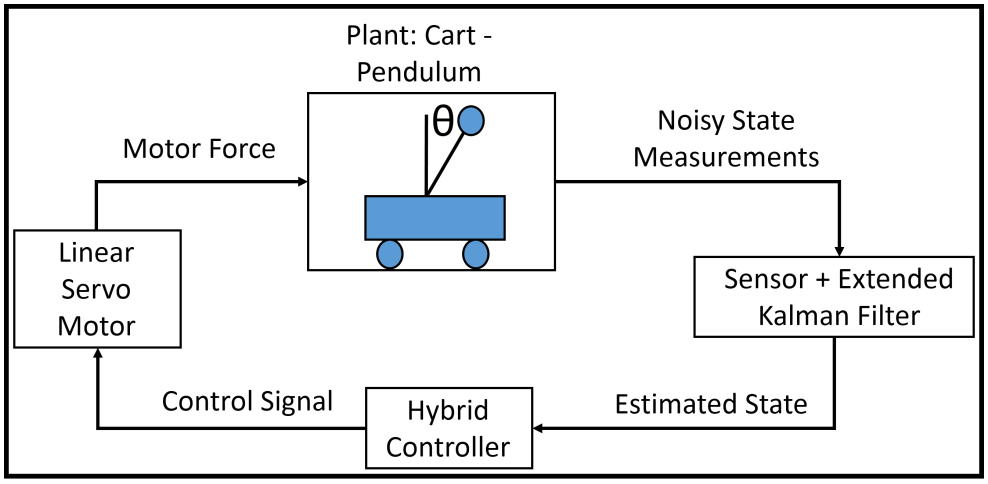


Figure 5.9: Inverted pendulum system mounted on a cart

Figure 5.9 shows the same block diagram of the inverted pendulum system described in details in the previous chapter.

5.5.2 Simulation Results

i) Pre-deployment training - The simulation experiments are conducted in Matlab on a Windows PC with 3.10 GHz processor. The pendulum-cart system is simulated with parameter values of $M = 3$ kg, $m = 0.2$ kg, $l = 0.31$ m and $g = 9.81$ m/s^2. The hybrid controller is designed to switch from the nonlinear controller to the state feedback controller for $|\theta| \leq 30^\circ$ where $\theta = 0^\circ$ is the vertically upright position. The linear feedback controller is designed to achieve 20% maximum overshoot and a 2% settling time of 2 seconds. The appropriate feedback gain K in (4.14) is determined using Ackermann's formula for pole placement. The measurement noise is modeled by a Gaussian distribution with mean 0 and standard deviation of 0.1. The normalized Gaussian network based actor critic method described in Section 5.1.4 is used as the RL controller. For each of the 4 dimensions $[\theta \quad \dot{\theta} \quad x \quad \dot{x}]$, the valid ranges are defined according to the maximum allow-

able values as: a) θ varies between $-\pi$ to $+\pi$, b) $|\dot{\theta}|_{\text{max}}$ is selected as 2.5π rad/s, c) the cart movement is restricted between $x_{\text{max}} = \pm 2.4$ and d) the maximum cart velocity is limited to $|\dot{x}|_{\text{max}} = 2$. The restrictions on state values are chosen to pose the balancing task as a constrained optimization problem. The 4-dimensional state space $[\theta \quad \dot{\theta} \quad x \quad \dot{x}]$ is divided into $10 \times 10 \times 3 \times 3$ grid for the critic basis functions. Similarly, the state-space is divided $10 \times 10 \times 10 \times 10$ grid for the actor implementation. Thus, the number of parameters in the critic weight $\mathbf{w}^c$ and actor weight $\mathbf{w}^a$ are $10 \times 10 \times 3 \times 3 = 900$ and $10 \times 10 \times 10 \times 10 = 10000$. The discount factor γ and the eligibility trace decay parameter λ for the value functions are both chosen as 0.5. The two learning rates η^c and η^a are selected as 0.8. The sigmoid function $s(.)$ implementation in (5.14) is defined from [139] as $s(x) = 2\frac{\tan^{-1}(\frac{\pi x}{2})}{\pi}$. The exploration term $\mathbf{n}(t)$ in (5.14) is a random perturbation of the policy with normally distributed zero-mean white noise with standard deviation $\sigma = 2$ as $\mathbf{n}(t) \sim \mathcal{N}[0, 2]$. Exploration is done every fifth step instead of every step [137] to allow for large exploratory actions while providing sufficient time to the AC controller to learn from the exploratory actions. The reward r is defined by the function $\bar{\rho}(\mathbf{x}) = \cos\theta$ where the maximum reward is obtained for $\theta = 0^\circ$. The actor-critic is trained along with the nominal controller as shown in Figure 5.2 through episodic experiments. Each episode is 20 seconds long and the balancing is initiated with the pendulum hanging vertically downwards. In presence of the nominal controller, the restrictions on the states are usually not violated and the pendulum is finally balanced in a vertical upright position. However, in occasions a learning episode is terminated due to violation of state restrictions caused by the exploration of actor policy and in such a case, the trial is rewarded with $r = -1$ to penalize the corresponding action taken.

After learning the nominal AC controller, the cart-pendulum system is simulated with 100 different initial angles uniformly sampled between $\theta = 180^\circ$ and $\theta = 10^\circ$ over 20 seconds of simulation. The sliding temporal window length is selected as 12 samples in order to accurately map the nonlinear dynamics [TDSC Paper]. The unified variable $\mathbf{s}(t)$

is selected as $\mathbf{s} = [y_1 \quad y_2 \quad u]$ where y_1 and y_2 are the two measurements in (4.15). The function $F_m(.)$ is selected as a linear function $F_m(\mathbf{s}) = y_1 + y_2 + u$. Thus, a MARS model with memory depth 12 is trained for implementing $F_c(.)$ such that $F_m(\mathbf{s})$ is predicted from the past 12 observations. The detection threshold of $|e_{nom}| = 0.05$ is established from the error distribution for fault-free error signal $e(t)$.

For the pre-deployment training exercise, the fault model is defined as actuator degradations and sensor data corruptions. The torque constant and the back emf constant of the linear motor is perturbed by a uniform distribution of 35% to create the systems with injected actuator faults. The sensor data in (4.15) is corrupted with a bias selected from a uniform distribution of 10% to generate systems with sensor faults. A total of 4000 Monte Carlo samples are created by sampling the fault universe of sensor and actuator faults. For all these systems, the error signal $e(t)$ is recorded along with the reference input $x_d(t)$ for $\Delta t = 2$ seconds. The TD error $\delta(t)$ is also recorded for the same duration and the magnitudes are summed up to generate a single value as an evaluation of the learning performance of the nominal AC controller for each fault-injected system. For the PNN-based scheme, a nonlinear regression model (MARS) is trained with the error signal $e(t)$ and reference input $x_d(t)$ to predict the perturbed parameter set - torque constant, back emf constant and the sensor output biases. Next, a PNN classifier is used to cluster the 4000 samples on the basis of the TD error magnitude generated previously into 3 clusters. For each of the clusters, one medoid sample is chosen to represent all other points in that cluster. For each of the medoid systems, a new optimal AC controller is trained by performing episodic experiments until the TD error $\delta(t)$ converges to zero, thus relearning how to balance the pendulum in presence of sensor and actuator errors. The weights of the 3 optimal AC controller for each cluster center are stored in memory. For the RNN-based scheme, 20 maximally separated samples are selected from the 4000 Monte Carlo samples such that the 4-dimensional fault space is reasonably sampled. For each of these 20 fault-injected systems, an optimal AC controller is learned and an RNN is trained with the time series data of $e(t)$ and $x_d(t)$ to

directly predict the AC weight parameters directly. As the diagnostic information is not predicted, the RNN requires less temporal data for training to target optimal AC weights. The clustering and the different AC controllers for the PNN-based scheme are shown in Figure 5.10.

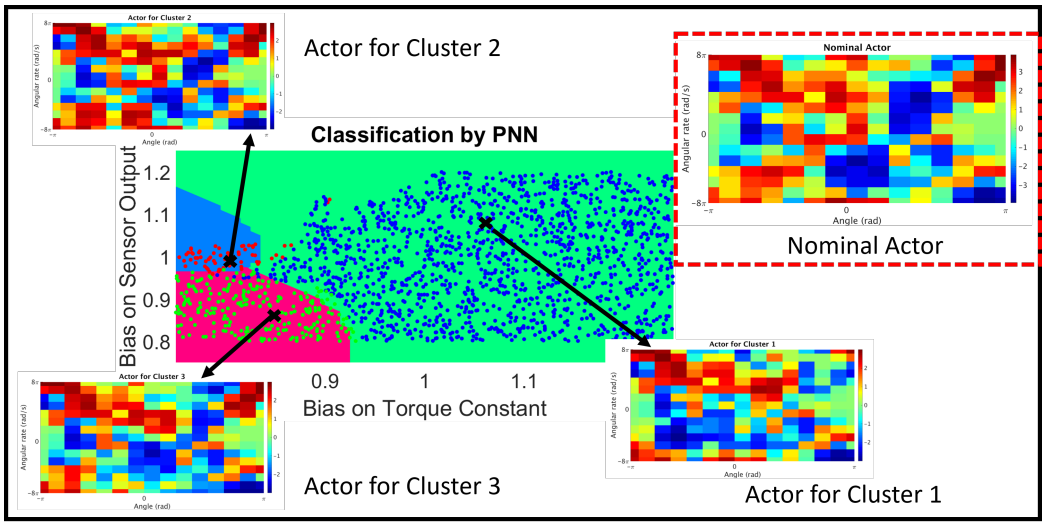


Figure 5.10: Classification in the parameter space by the PNN and subsequent generation of representative optimal AC controllers for each of the cluster centers

The subfigure in the right top inset of Figure 5.10 indicates the nominal AC controller trained for the fault-free system. The different colors demonstrate the varying control force u_a between [-3,+3] imparted in addition to the nominal control u_c at different state space locations. The original state space is 4-dimensional and for visual demonstration, this figure is a projection of actor policy with changing θ and $\dot{\theta}$ on $x - \dot{x}$ plane. The classification plot in Figure 5.10 is also a projection from the 4-dimensional fault parameter space to a 2-dimensional subspace where the x and y axes show the bias on torque constant and bias on the sensor output y_1. The three optimal actor-critic controllers depicted are relearned on the fault-injected systems represented by the cluster medoids and they are different from the nominal one as seen in Figure 5.10.

ii) Post-deployment experiments - For demonstration of the real-time accelerated learning procedure, fault injection experiments are performed. Firstly, the necessity of a self-

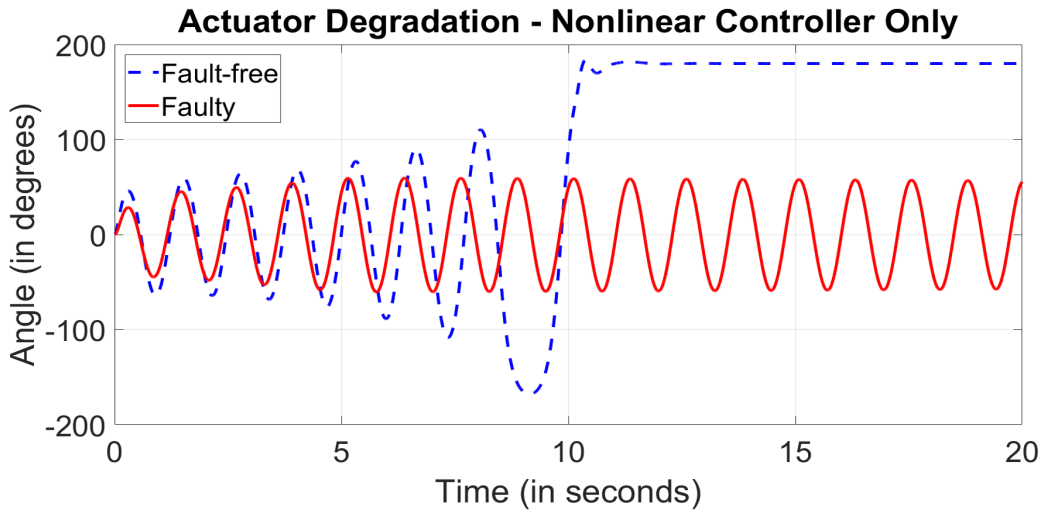


Figure 5.11: This shows the need of a self-learning assisted controller design where a nominally designed controller is unable to meet system functionality under actuator degradation

learning assisted controller design is demonstrated in Figure 5.11. Only the hybrid controller is used and the augmentation $u_a(t)$ from the AC controller is disabled. In the fault-free system, the nominal controller is able to balance the pendulum vertically at $t = 10.83$ seconds. An actuator fault of 20% degradation in the torque constant of the motor is gradually injected from $t = 0$ and the maximum degradation of 20% is complete at $t = 4$ seconds according to (5.16) (the fault is inserted at a high rate for illustrative purposes). The red curve shows the angle of the pendulum in presence of the fault. It is seen that the hybrid controller fails to balance the pendulum in presence of the actuator fault and a self-learning module is necessary to supplement the nominal controller for providing correction capabilities.

Figure 5.12 demonstrates the learning acceleration enabled by ALERA-PNN and ALERA-RNN in comparison to the nominal learning of the augmented controller. The same 20% actuator degradation is injected as mentioned above. In the fault-free case, the pendulum is vertically balanced at $t = 10.83$ seconds. In presence of the actuator degradation, the pendulum is vertically balanced at $t = 28.8$ seconds without the assistance of the learning accelerator module. With the PNN-based learning accelerator enabled with 3 clusters used for the PNN training, the pendulum gets balanced at $t = 13.8$

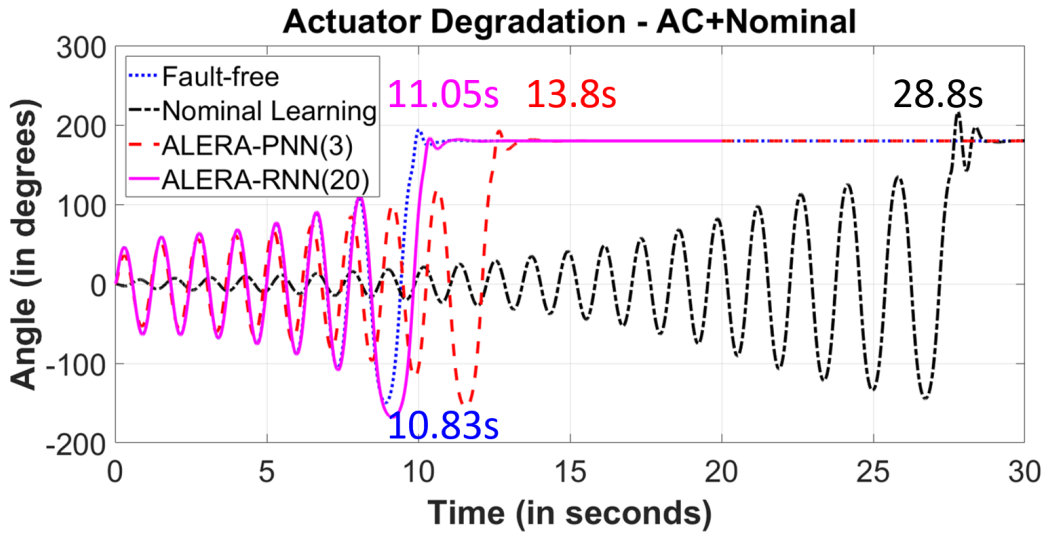


Figure 5.12: Demonstration of learning acceleration of ALERA in presence of actuator degradations

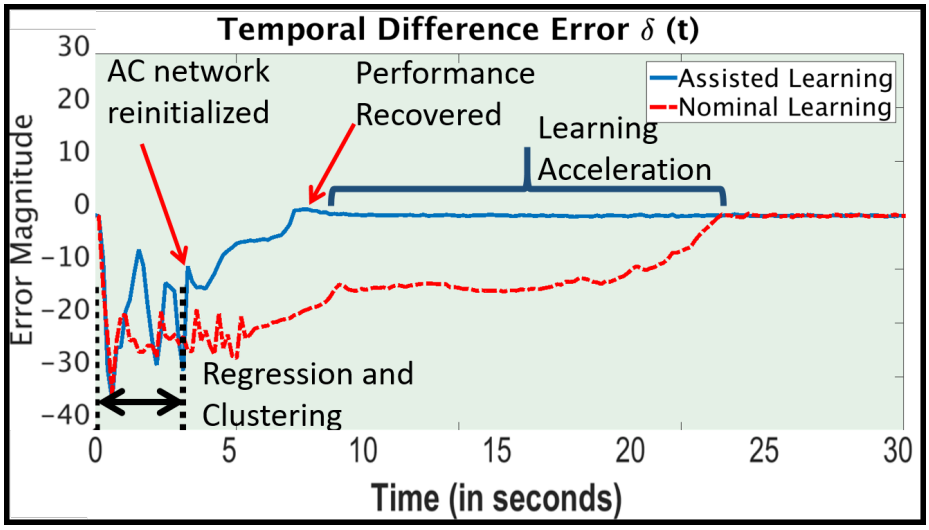


Figure 5.13: Comparison of TD error between PNN-based learning enabled and disabled schemes

seconds, thus accomplishing the desired functionality faster than the nominal learning. However, the RNN-based learning scheme balances the pendulum at $t = 11.05$s providing significantly low latency in adapting to the altered circumstances. The accelerated learning performance is demonstrated in Figure 5.13. The TD error $\delta(t)$ is plotted against time for both the nominal and accelerated learning scheme for the PNN. The TD error deviates from zero at $t = 0$ due to injection of the actuator fault. The AC network with

pre-learned weight parameters is suboptimal in presence of the fault and the TD error starts guiding the gradient based reinforcement learning algorithm in both cases. For the assisted learning scheme, from $t = 2$s to $t = 4$s, the MPCS based error signal $e(t)$ (not shown in figure) that shows non-zero values indicating the presence of errors, is recorded. From $t = 4$s to $t = 4.6$s, the regression model maps the observed error signal $e(t)$ and reference input $x_d(t)$ to the parameter space and then the pre-trained PNN assigns cluster label 2 to the predicted parameter set. At $t = 4.6$s, the actor and critic weights are initialized with the stored value for cluster 2 shown in Figure 5.10 and the TD error instantly decreases in magnitude from -31.4 to -13.8, thus emphasizing the acceleration in discovering the optimal parameters through learning. Henceforth, the AC learning in the acceleration-enabled scheme converges fast at $t = 7.2$s and manages to balance the pendulum at $t = 13.8$s while the nominal AC controller completes its learning at $t = 23.7$s and balances the pendulum at $t = 28.8$s. Defining the time to achieve the same application level functionality through nominal and assisted schemes as t_n and t_a respectively, the application level 'learning acceleration' ψ is defined as $\psi = \frac{t_n}{t_a}$. Thus, the acceleration ψ for ALERA-PNN is $\psi = \frac{28.8-10.83}{13.8-10.83} = \frac{17.97}{2.97} = 6.05$. Similarly, the learning acceleration ψ for ALERA-RNN is given as $\psi = \frac{28.8-10.83}{11.05-10.83} = \frac{17.97}{0.22} = 81.68$. Thus, the learning acceleration of ALERA-PNN and ALERA-RNN provides a 6.05X and 81.68X boost for application level compensation respectively.

iii) Analysis and discussion - The learning acceleration ψ depends on how close the optimal actor-critic weights proposed by the accelerator module are to the actual optimal values achieved after convergence of learning. As discussed in Section 5.3.1, the cluster medoid is a representation for all systems in that cluster for which the learning behavior of the nominal AC is similar. With the increase in the number of clusters, the disparity between a system close to any cluster boundary and the medoid decreases and the potential for learning acceleration increases. The performance variation with the increase in cluster count is depicted in Figure 5.14 along with the upper and lower bounds. 5000 different

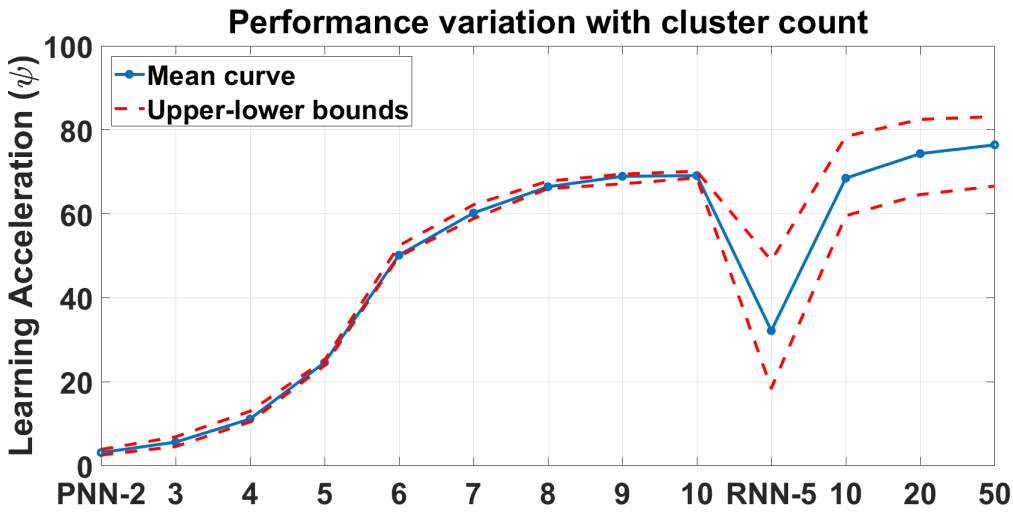


Figure 5.14: Comparison of learning acceleration ψ between PNN and RNN schemes with PNN variable as number of clusters and RNN variable as number of training sampels

fault-injected systems are created by exhaustive sampling of the 4-dimensional fault space described previously. For each of those systems, the ALERA-enabled learning acceleration is applied with the PNN trained with different number of clusters in pre-deployment phase. The mean acceleration ψ achieved along with the maximum and minimum for each cluster count is plotted in Figure 5.14. It is seen that ψ increases from 3X to 70X with increase in the cluster count from 2 to 10. However, the trend shown in Figure 5.14 illustrates that the rate of increase of ψ is initially high and it starts tapering off after cluster count of 6 showing diminishing benefits with increasing clusters. After 10 clusters, the acceleration of the RNN-scheme is presented. The RNN interpolates the optimal AC weights depending on the training received. An RNN trained with 5 data samples provides a mean ψ of 32X and the acceleration grows up to 80X for training with 50 data samples. However, due to the interpolation by the RNN based on a few training cases the performance varies widely with the maximum and minimum bounds gradually reducing with the increasing number of training data samples.

The memory overhead required along with the machine time consumed for pre-deployment training is shown in Figure 5.15. It is seen that the required memory

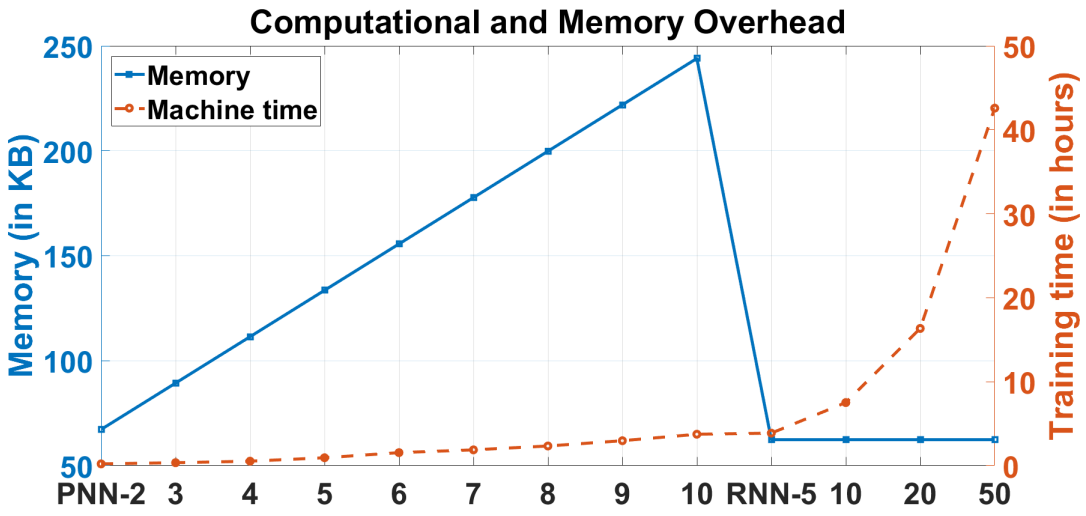


Figure 5.15: Memory overhead and machine time required for pre-deployment training in the PNN and RNN based schemes

overhead increases linearly with the number of clusters for the PNN-based scheme. The depicted memory overhead indicates the space required to store the learned AC parameters corresponding to each cluster center along with the trained PNN classifier and the regression model. The RNN-based scheme requires significantly low memory since only the trained RNN is stored without separately storing any individual AC parameters. The memory requirement of the RNN is fixed with the number of training data as only the trained network is stored. Though it may seem from Figure 5.14 and 5.15 that the RNN-based scheme provides high learning acceleration ψ with small memory requirement, the pre-deployment simulation time necessary to properly train the RNN is quite large in comparison to the PNN-based schemes. As seen in Figure 5.15, the RNN-based scheme takes significantly longer time for pre-deployment training than that of PNN-based scheme. For example, the RNN scheme with only 5 training data samples takes 3.88 hours of machine time in comparison with 3.75 hours in the PNN scheme with 10 clusters. This significant machine time requirement is due to two reasons: a) the training data of the RNN requires the convergence of AC learning for an increased number of systems and b) since the RNN is trained to directly predict the AC parameters (10900

output data points) from the time-series data of the error signal $e(t)$ and reference input (4000 input data points), the training procedure is significantly slower than that of a PNN. It is seen in Figure 5.15 that the machine time requirement for the RNN-scheme increases exponentially with the increase in number of training data and quickly becomes infeasible due to the huge volatile memory requirement during the training of the RNN. Thus, for applications where pre-deployment simulation is expensive, an RNN-based scheme may not be feasible despite providing better acceleration than a PNN-based clustering scheme.

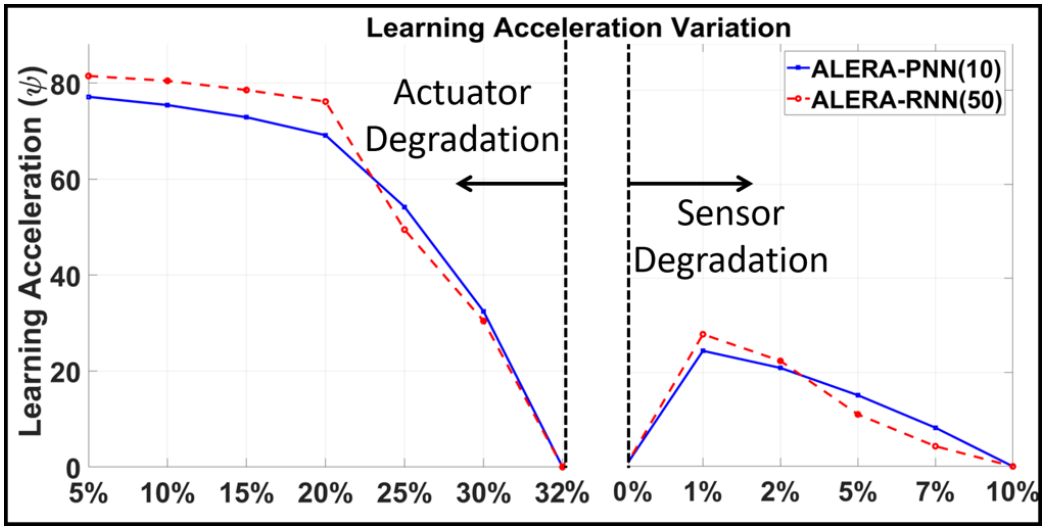


Figure 5.16: Variation of learning acceleration ψ for ALERA-PNN and ALERA-RNN with different degrees of sensor and actuator degradation

Figure 5.16 shows the variation of learning acceleration ψ for the PNN and RNN schemes in presence of actuator and sensor degradations. It is seen that sensor faults are more difficult to correct than actuator errors. The data shown is for the ALERA-PNN with 10 clusters and ALERA-RNN with 50 data samples for training. The left portion of Figure 5.16 represents actuator errors ranging between 5%-32% while the right portion denotes sensor degradations with errors ranging between 1%-7%. It is seen that as the magnitude of injected faults increases, the learning boost achieved by using ALERA's acceleration reduces for both sensor and actuator errors. For test case I, ψ is shown to be zero at actuator and sensor degradations of 32% and 10% respectively to indicate that the AC controller

fails to balance the pendulum and functionality is never achieved with fault magnitudes equal to or greater than that. This is explained from the fact that the AC controller is used to augment the nominal controller by providing a supplementary action u_a to the nominal control u_c. However, the combined action $u = u_c + u_a$ has to satisfy the physical limitation of applied control u, defined by u_{max}. This is accomplished by choosing individual maximum restrictions $u_{c(max)}$ and $u_{a(max)}$ for the nominal and AC controller in the design phase. The ratio $\phi = \frac{u_{a(max)}}{u_{max}}$ determines the extent of impact that the AC controller exercises during the reconfiguration. $\phi = 0$ indicates that the nominal controller is solely responsible for providing adequate control with $u_a = 0$ whereas $\phi = 1$ represents that the AC controller is completely responsible with $u_c = 0$. The linear and logarithmic variation of ψ with ϕ for ALERA-PNN with 10 clusters under different actuator and sensor errors is shown in Figures 5.17 and 5.18.

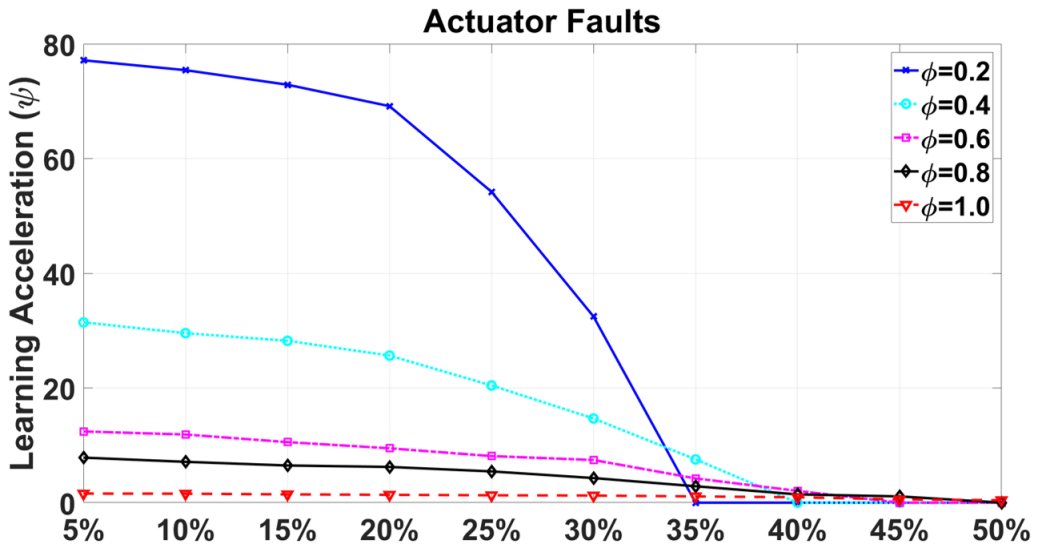


Figure 5.17: Linear variation of learning acceleration ψ with ϕ under different magnitude of injected actuator errors

Figures 5.17 and 5.18 demonstrate the variation of learning potential ψ with actuator error magnitude under 5 different values of ϕ - 0.2, 0.4, 0.6, 0.8 and 1. The key observations from this plot are:

1. As ϕ increases from 0.2 to 1.0, the contribution of the RL controller to the total

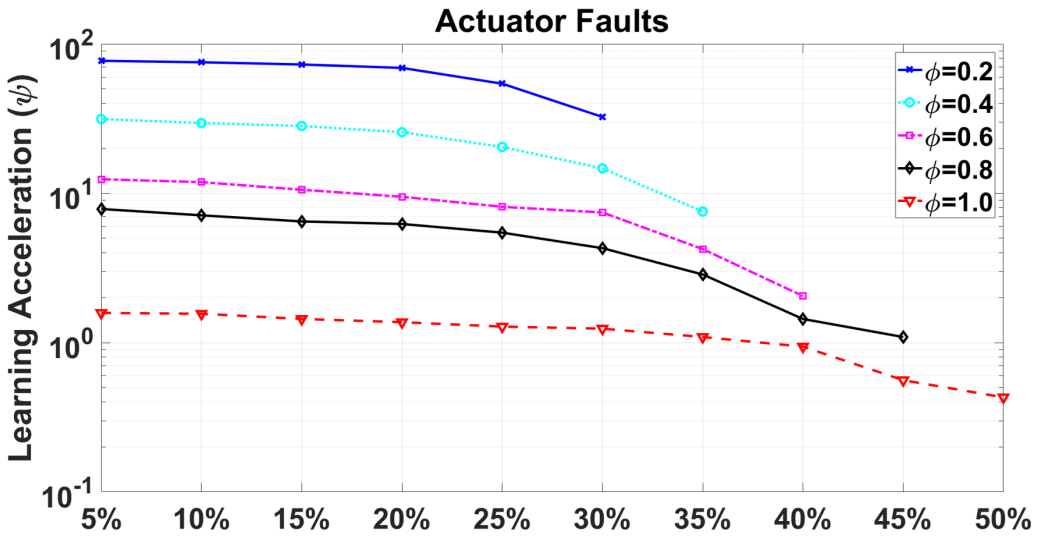


Figure 5.18: Logarithmic variation of learning acceleration ψ with ϕ under different magnitude of injected actuator errors

control applied to the plant increases. This enhances the extent of self-learning capabilities of the RL controller since it forms a higher portion of the applied control. As a result, the AC controller is able to tolerate and correct higher magnitude of injected actuator errors as seen from Figure 5.18. With $\phi = 0.2$, the maximum actuator error that can be corrected with restoration of system functionality is 32% whereas with $\phi = 1.0$, the maximum actuator error that can be corrected is 51%.

2. With the increase of ϕ, the learning acceleration ψ decreases. For example, with the same 5% actuator error, the learning boost is $\approx$ 80x with $\phi = 0.2$ whereas with $\phi = 0.8$ the learning acceleration is only 8x. This is explained from the fact that as ϕ increases, the AC controller has to contribute to the bulk of the applied action and has to perform significant learning resulting in increased time for performance recovery, thus effectively reducing the learning acceleration achieved.

This analysis demonstrates that there is a trade-off in the choice of the ratio ϕ. A lower value of ϕ will provide self-recovery faster but will correct less errors whereas a higher value of ϕ will recover from more severe errors but the adaptation will be slow.

5.6 Test Case II: Brake-by-Wire System

5.6.1 System Description

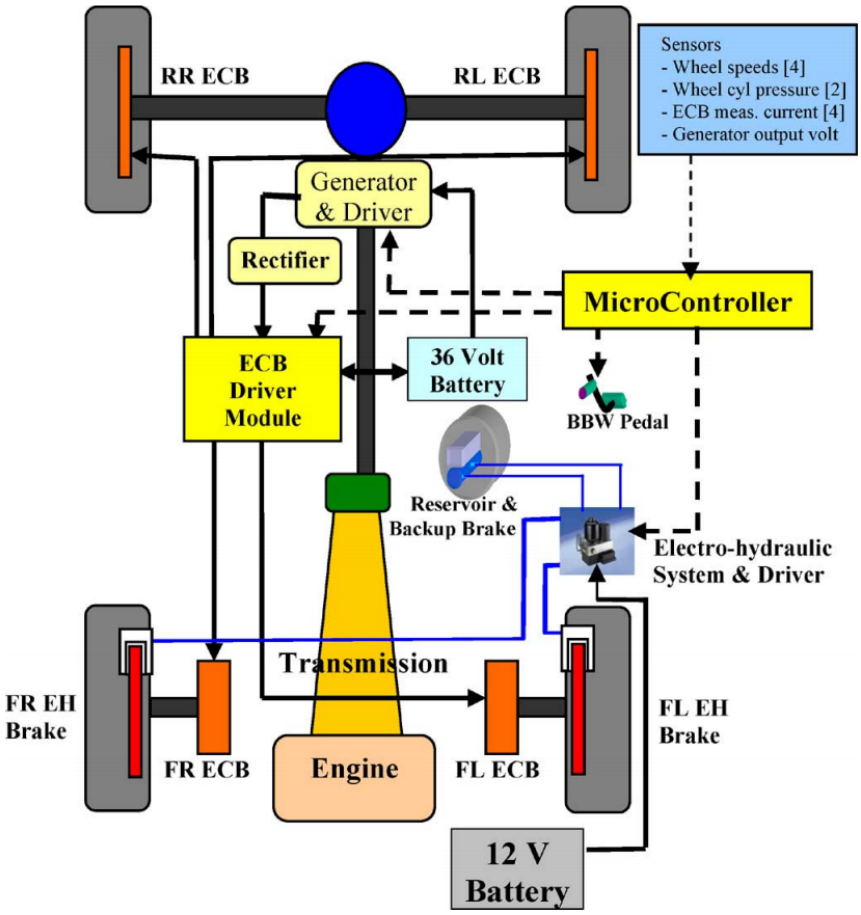


Figure 5.19: A brake-by-wire system for a rear-wheel drive vehicle

Figure 5.19 shows the schematic block diagram of the brake-by-wire system described in details in the previous chapter.

Similar to the previous test case, a nominal AC controller is trained in addition to the sliding mode controller described above with 500 fault-free episodic simulations of the vehicle dynamics with different applied pedal forces. The AC controller is provided a positive reward if the vehicle deceleration $\dot{v}_x$ matches the reference input deceleration and penalized with a negative reward proportional to the tracking error. The reward function generates the maximum negative reward if the angular speed of any wheel is less than the

equivalent linear velocity of the vehicle, indicating a potential wheel lock-up condition. The range of angular speeds for each wheel is defined as 0-100 rad/s with a maximum linear velocity of 100 mph. The 5-dimensional state space is divided into grids of 10 for each dimension, thus creating $10 \times 10 \times 10 \times 10 \times 10 = 10^5$ parameters for each of the actor and critic. However, in this test case, the actor and critic parameter matrices are sparse since the wheel speeds ω_is and the vehicle speed v_x are tightly coupled and it's quite impossible for the wheels to have completely different angular speeds. The parameter choices of the reinforcement learning and the sigmoid function selection are the same as in the previous test case.

5.6.2 Simulation Results

Table 5.1: Parameter values for BBW system

Parameter	**Values**
M	1300 kg
R	0.4 m
I_w	0.6 kg/m^2
F_z	2000 N
α_{si}	4.5
κ_{th}	0.2
η	0.8
ϕ	1.5

i) Pre-deployment training - The values of the different parameters for the BBW system are shown in Table 5.1. The MARS prediction function $F_c(.)$ in the error detection module is trained with braking data for 1000 different initial vehicle speeds ranging between 50 mph and 100 mph with the temporal sliding window length chosen as $T_p = 6$ samples [TDSC paper]. The function F_m is selected to generate the sum of wheel speeds. A Gaussian distributed measurement noise with mean 0 and standard deviation of 2 rad/s is assumed while measuring the angular speed ω_i of each wheel. For training the learning accelerator module, the fault model is defined as perturbations in the wheel speed readings (sensor errors) and corruption of the ECB signal (actuator errors) by altering the applied

braking torque by 20%. A total of 1000 fault-injection experiments are created and a PNN classification with 4 clusters is adopted, followed by training of optimal AC controller for each cluster medoid. For the RNN-based scheme, 50 different fault-injected samples are created and an optimal AC controller is learned for each of these systems. The RNN is trained with the time-series data of $e(t)$ for all such systems to directly predict the 10^5 AC parameters.

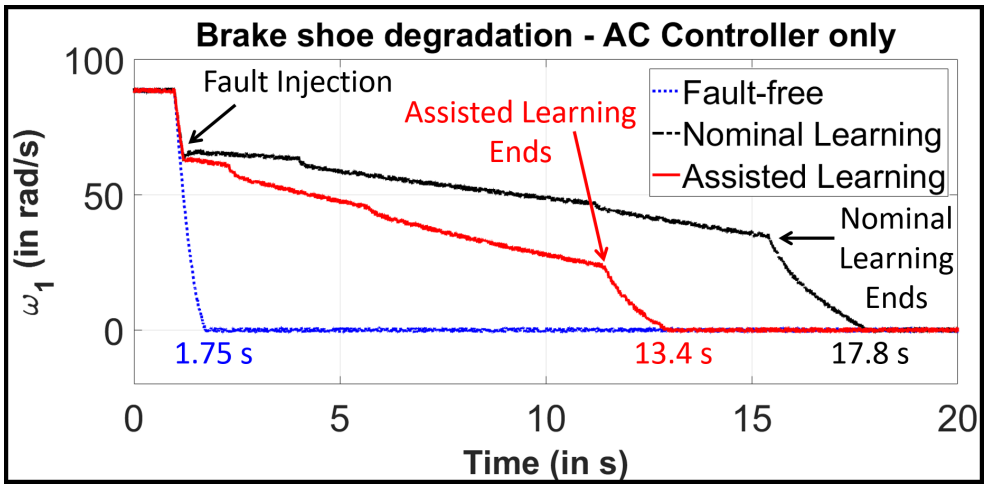


Figure 5.20: Implementing an AC network as the sole controller instead of using in an augmented setting of ALERA provides 1-2X boost in the learning speed and fails to achieve the desired braking performance

ii) Post-deployment experiments - Figure 5.20 illustrates a braking situation with the AC network as the sole controller instead of the augmented scheme (nominal controller+AC network) as shown in Figure 5.2, in presence of a degraded brake shoe. The braking is initiated at $t = 1$s from an initial wheel angular velocity of $\omega = 90$ rad/s, equivalent to linear velocity of 90 mph for a wheel radius of 0.4 m. A brake shoe degradation of 20% is injected at $t = 1.2$s, thus reducing the applied braking torque. In the absence of any fault, the braking event is completed at $t = 1.75s$ with complete vehicle stoppage. The two different curves represent systems with and without learning acceleration enabled. Similar to test case 1, the check error signal $e(t)$ (not shown in Figure) is recorded from $t = 1.2$s to $t = 2.2$s and the trained regression model in the PNN-based scheme estimates a perturbed parameter set in the fault universe from the

transient waveform. The AC weights are initialized from the stored data corresponding to the cluster medoid predicted by the PNN at $t = 2.5$s and learning is resumed from the new weights and the AC converges at $t = 11.42$s, thus partially recovering the braking performance as seen. On the contrary, the unassisted AC controller completes its relearning at $t = 15.4$s. From an application level point of view, the stoppage of vehicle ($v_x = 0$) is completed at 13.4s in the assisted learning scheme compared to 17.8s in the nominal learning scheme. Since the fault-free stopping is completed at $t = 1.75$s, a learning acceleration of $\psi = \frac{17.8-1.75}{13.4-1.75} = \frac{16.05}{11.65} = 1.37$ is achieved. Thus, the learning acceleration ψ is only 1.37X along with an extra stopping distance of 216.8 ft, thus endangering the vehicle and failing to achieve the functionality of high-speed braking.

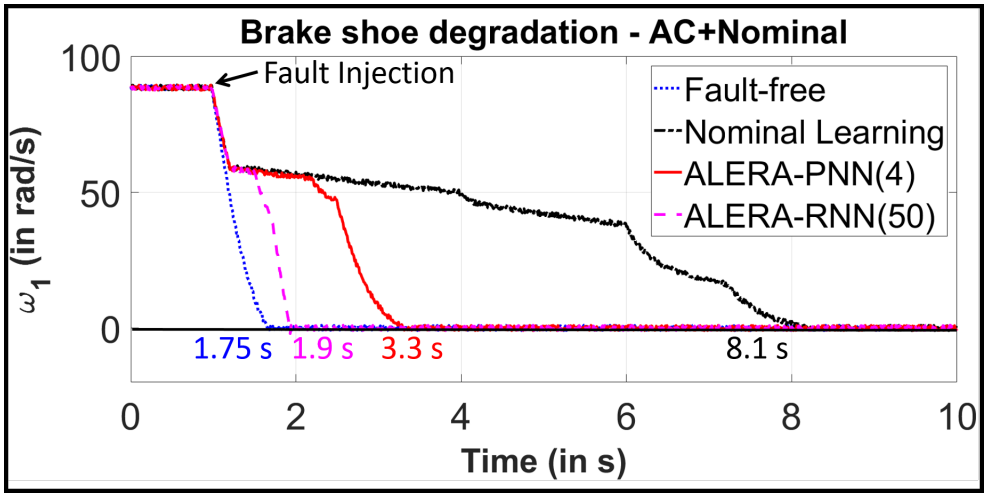


Figure 5.21: The augmented scheme under ALERA achieves the functionality in presence of 20% of actuator degradation with an additional stopping distance of 10 ft. The RNN-based scheme provides a 43X acceleration while the PNN-based scheme delivers 4.12X boost

Figure 5.21 shows the braking performance of the augmented scheme in ALERA where the AC network is used to supplement the performance of the sliding-mode controller. As shown in Figure 5.20, the fault-free braking is completed at $t = 1.75$s. After the actuator injection, the nonzero TD error $\delta(t)$ (not shown in Figure) guides the RL learning in the unassisted case and the learning completes at 7.2s and completing the vehicle stoppage at 8.1s. In comparison, the ALERA-PNN scheme with 4 clusters completes the learning at

2.5s and stops the vehicle at 3.3s, thus providing a learning acceleration of $\psi = 4.12$. The ALERA-RNN scheme trained with 50 pre-deployment training data completes the learning at 1.47s and stops the vehicle at 1.9s providing a boost of 43X.

5.7 Test Case III: Self-balancing Robot

5.7.1 System Description

To demonstrate the benefits of the proposed approach, the self-balancing robot described in the previous chapter is considered. The control algorithm along with the RL controller and its concomitant modules are implemented on the Atmel microcontroller with the Arduino programming environment. The actuator fault is injected as a corruption of the motor drive signal from the two Texas Instruments DRV8838 motor drivers that modify the resulting torque generated in the two brushed DC motors. The nominal self-balancing algorithm is a hybrid implementation of a nonlinear swing-up controller and a PID controller. The nonlinear swing-up controller provides PWM motor drive signals in alternate directions, thus creating a jerking motion to bring the robot close to vertical, after which the PID controller takes over for fine balancing around the equilibrium position of $\theta = 0^{\circ}$. The maximum motor speed allowable while attempting to balance the robot is 300 RPM.

5.7.2 Hardware Results

The actuator fault model involves changing the motor drive signal computed in the control algorithm. This motor drive signal is transmitted from the microcontroller to the motor drivers that drive the motors with appropriate current. Corrupting the motor drive signal after the control computation alters the torque generated by the motors and affects the system operation. The effect of 20% actuator fault is illustrated in Figure 5.22.

Figure 5.22 demonstrates that without error, the robot manages to self-balance in 3s and stays upright with the angle θ around 0°. With the injected 20% error, the control algorithm attempts to swing up the robot and balance itself but fails to accomplish this task. The robot

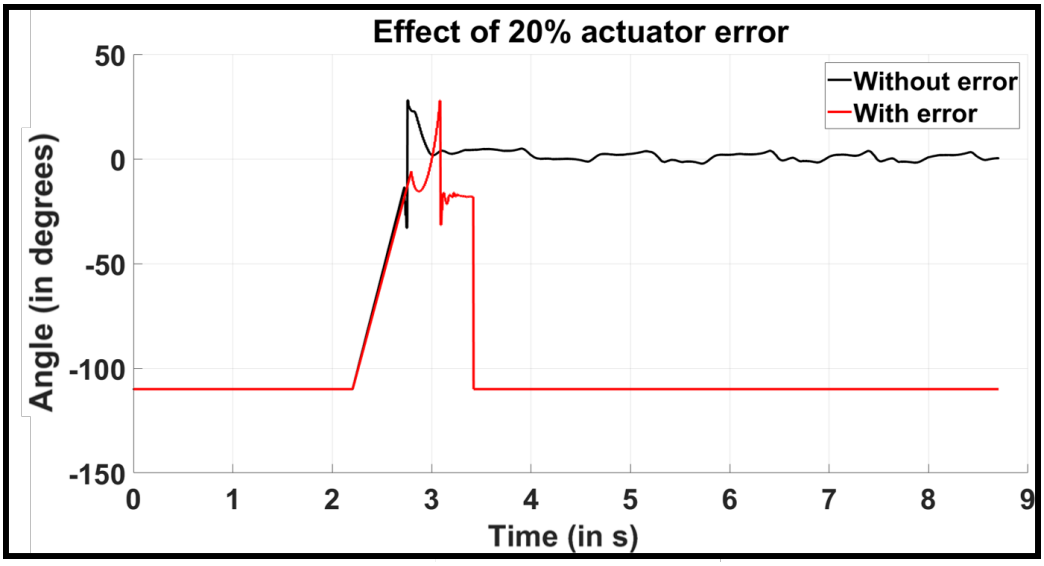


Figure 5.22: With 20% injected actuator error, the robot fails to stand up and balance

falls back to the lying down position with angle $\theta = 110^{\circ}$ at 3.4s and never manages to rise up and balance itself.

To implement ALERA in the hardware platform of the balancing robot, the nominal controller is augmented with a reinforcement learning controller with a ratio $\phi = 0.33$. The motor drive signal from the nominal controller is restricted at a maximum value of 200 RPM while the RL controller's maximum value is fixed at 100 RPM. For the checking function implementation, the MARS model trained using the data from 100 balancing experiments is utilized and a polynomial implementation of the MARS model is adopted. This polynomial function is used as the checking function for the error detection module in ALERA. For the RL controller implementation, the 4 states of the system (angle, angular velocity, position, linear velocity) are discretized in $4 \times 4 \times 4 \times 4 = 256$ grid locations resulting in 256 actor and 256 critic parameter values. 50 hardware episodic experiments are conducted such that the RL controller learns how to supplement the nominal controller in the balancing task. The 512 actor and critic parameter values are stored as nominal RL controller parameters.

The sensor fault model is implemented by changing the DOF data from the accelerometer and the gyroscope along with corruption of the quadrature encoder readings. As de-

scribed previously, the actuator fault model involves changing the motor drive signals. For practical feasibility purpose, the fault model space is restricted to data corruptions in 3 measurements - i) the z-axis data from the accelerometer (alters the angular velocity), ii) the quadrature encoder readings signifying the rpm of the motors (alters the linear speed) and iii) the motor drive signals (alters the applied torque to the motor). 40 different points are sampled uniformly in this 3-dimensional space with a maximum deviation of 25% of the nominal values. For all these 40 configurations, the robot fails to balance using the nominal RL parameters. As described in Section 5.3.1, the error signal is recorded for 1 second and a regression model is trained in Matlab to predict the fault parameters that represent the bias in the measurement and actuation values for the fault models described above. A clustering is performed based on the TD error data recorded for 1s from the nominal RL balancing effort. The 40 points are clustered in the fault parameter space based on the TD error data into 5 clusters and 5 different medoid points are chosen to represent each of the clusters. For each of these 5 fault configurations, the learning of the RL controller is allowed such that it relearns to balance the robot in presence of the injected errors. After the 5 RL controllers converge, the resulting AC parameter values are recorded and stored.

The regression model for predicting the parameter values and the classifier required to cluster these points are impossible to implement in the Arduino microcontroller due to lack of processing power and on-board memory required to implement these algorithms. Hence, as a simplified alternative, a ℓ_2-norm based matching algorithm is adopted. The 256 parameter values for the 5 optimal AC controllers along with the error signals for the fault sets they represent are stored in the flash memory of the microcontroller. In actual operation, the ℓ_2-norm of the recorded error signal is compared with the stored error signals and the AC parameters for the closest matching error signal is chosen as the bootstrapped version of the RL controller. The balancing efforts with disabling and enabling of ALERA are shown in Figure 5.23.

In the ALERA disabled scheme, the learning of the RL controller starts from the nomi-

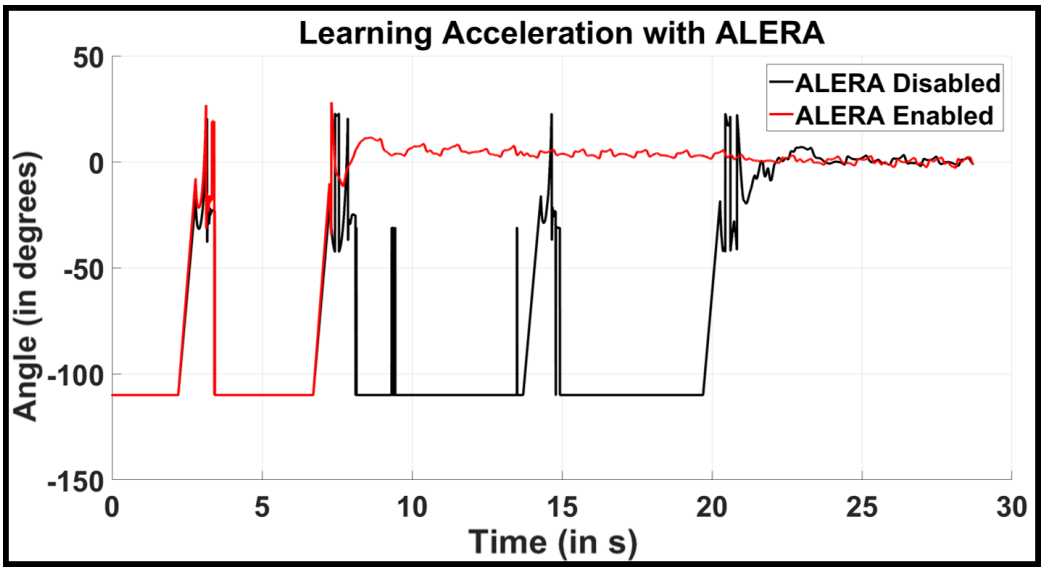


Figure 5.23: Demonstration of the accelerated learning enabled by ALERA: the robot is balanced significantly faster than the nominal learning procedure of the RL controller by bootstrapping the learning using the error signal

nally learned values and through repeated episodic tasks, the learning converges effectively relearning to balance the robot as seen in Figure 5.23. In the ALERA enabled scheme, in the first effort as the robot fails to balance itself, the error signal is recorded for 1s and the matching algorithm bootstraps the actor-critic learning by reinitializing the parameter values from the stored repository. It is seen in Figure 5.23 that the robot gets balanced in the 2nd attempt after reinitialization of the actor-critic parameter values. The nominal fault-free balancing is accomplished at 3s while the ALERA-disabled scheme balances it at 23.8s after 4 episodic experiments. The ALERA-enabled scheme manages to balance the robot at 9.3s. Thus the effective learning acceleration in this experiment is $\psi = \frac{23.8-3}{9.3-3} = \frac{20.8}{6.3} = 3.3$. This demonstrates that a learning boost of 3.3x is achieved in the actual hardware experiments using the assisted learning scheme of ALERA.

5.8 Summary

In this work, a real-time self-learning methodology for error correction and control law adaptation in nonlinear control systems is presented. Reinforcement learning algorithm is

proposed as an augmentation to a classical nonlinear controller to provide self-learning capabilities. The error detection methodology presented in the previous chapter is exploited to bootstrap the learning process, thus accelerating the recovery of system performance. Simulation results on two nonlinear test cases as well as hardware results strongly demonstrate the effectiveness and benefits of the proposed research. The different analysis of design choices present a practical trade-off in deciding the appropriate parameters of the scheme.

CHAPTER 6
DETECTION, DIAGNOSIS AND MITIGATION OF SECURITY ATTACKS IN CYBER-PHYSICAL SYSTEMS

6.1 Introduction

In this work, we focus on electric power grid as an example of cyber-physical systems. A power grid consists of four major components: i) *generation* produces the electric energy from different sources such as fossil fuels, hydro-electric forces, nuclear reactions and in recent times, from wind, solar and tidal forces; ii) *transmission* transfers the generated power through a very high voltage infrastructure; iii) *distribution* delivers the power for consumption; and iv) *consumption* uses the electrical energy for industrial, commercial and residential purposes. The smart grid integrates communication and information technologies with the physical processes to provide better "situational awareness" for efficient and intelligent demand handling methods, enabling a drastic cost reduction for both power generation and consumption. However, such reliance on communication networks to transmit measurements and control packets increases the possibility of adversarial attacks. Cyber-physical security involves both information security mechanisms and system-theoretic methods. Information security methods such as authentication, access control, encryption protocols and data integrity are insufficient since these techniques do not exploit the compatibility of system data with the underlying physical process and control architecture. For example, the Stuxnet attack[54–56] that infected an Iranian nuclear-enrichment plant in 2010, corrupted the centrifuges' measurements to indicate routine operation while simultaneously modifying the centrifuges' actuation signals to enforce complete destruction. This illustrates the necessity of a integrated procedure combining cyber and physical protection methods to ensure cyber-physical security. In this work, we propose physical security methods from a

control-theoretic perspective for attack detection and mitigation.

Modern smart grids are controlled by two primary mechanisms - i) a supervisory control and data acquisition (SCADA) system; and ii) an energy management system (EMS). The SCADA system collects operational data from an extensive sensor network dispersed over the entire power grid and performs state estimation, contingency analysis and optimal power flow computation. The EMS processes the SCADA data for generation control and scheduling applications to satisfy the power requirements of the network in an optimal manner. The cyber-infrastructure forming the backbone of SCADA and EMS is vulnerable from security attacks carefully designed to evade common detection schemes of such systems. For state estimation, continuous power flow measurements along network branches are taken by remote terminal units (RTUs) and relayed back to the control center. In a power grid, the state variables are bus voltage angles and magnitudes. The measurements include active and reactive power flows along the branches and active and reactive power injections at the buses. Power system state estimation (PSSE) methods previously employed static estimation techniques such as the weighted least squares (WLS)[159–161] that rely on data received intermittently from the RTUs (typically 2-4s interval). With the advent of phasor measurement units (PMUs) that measure the temporal voltage and current profiles using a common synchronization time source, dynamic state estimation techniques[162–166] have been proposed that incorporate the PMU measurements in the WLS state estimation. However, the heavy bandwidth requirement and high initial capital cost have prevented the pervasive deployment of PMUs with ongoing research targeting optimal PMU placements [167, 168] and data compression techniques[169]. Typically, PSSE methods perform state estimation from the measurements only instead of incorporating the knowledge of system dynamics because a sufficiently accurate dynamic model of the entire network is difficult to obtain and even if such a model exists, the required computational effort renders the exercise infeasible. The malicious attack injections exploit these vulnerabilities and are crafted to bypass existing detection capabilities, thus misguiding the state estimation meth-

ods and disrupting regular system operation. Inspired from the use of encoded check states for error detection in linear [125] and nonlinear [170] control systems, this work proposes a comprehensive framework for low latency attack detection, diagnosis and recovery for security attacks on these power grids instead of separate techniques for each attack type as reported in all previous work.

Primary Contributions: The primary contributions and benefits of this research in comparison to prior work are:

i) Low latency detection - The proposed scheme uses encoded check states for detection of security attacks in power grids with low latency.

ii) Resilience to counterattacks - It is demonstrated that the proposed methodology is highly resilient to adversarial counter-attacks. The attacker needs complete knowledge of power grid dynamics to be able to reverse-engineer the detection mechanism and a simple scheme is mentioned to thwart any brute-force approaches. To the best of the authors' knowledge, no other work has ever undertaken an effort to consider adversarial counter-strategies to undermine the efficacy of any detection scheme.

iii) Low latency diagnosis and recovery - We propose check state assisted diagnosis and recovery methodologies with significantly reduced latency in comparison to previous works.

iv) Low complexity - The complexity of the proposed diagnosis algorithm is $\mathcal{O}(n)$ where n is the number of states in the system. This is significantly smaller than $\mathcal{O}(n^k)$ reported in prior work, where k is the number of sensors/actuators under attack.

v) Extensive simulation - Finally, we emphasize the impact of our proposed research with simulation of IEEE benchmark power systems and massive European high voltage distributed power grids. Previous works have not considered how the competing schemes scale up with the size of the power grids.

The rest of the chapter is organized as follows. Section 6.2 discusses the related work on security attacks in cyber-physical systems and highlights the primary contributions of

this research. In section 6.3 we provide an overview of a cyber-physical system and mathematically formulate the problem of security attack detection, diagnosis and mitigation. Section 6.4 discusses the preliminary ideas of state estimation in power grids. The different security attack models considered in this work are described in Section 6.5. Sections 6.6 and 6.7 details the proposed detection and mitigation schemes. Simulation results are presented in Section 6.8.

6.2 Prior Work

In [171], Amin et al. first define *deception* and *denial-of-service* attacks against any networked control system. Deception attacks corrupt the integrity of control and measurement packets by altering the sensor and actuator communications. Denial-of-service attacks, instead, compromise the resource availability by communication jamming. Liu et al.[172] introduce *false data injection* (FDI) attacks against DC state estimators for power grids in 2009. These are stealthy deception attacks in which an attacker exploits the power system configuration to inject sensor errors that bypass traditional detection techniques. Later on, Hug and Giampapa[173] focus on FDI attacks on AC state estimation and discuss the attack strategies from an adversarial perspective. Recent works[174–176] have further expanded the discussion on FDI attacks against AC state estimation. In [177, 178] Sinopoli et al. discuss another type of attack known as *replay attack* against control systems. These attacks are executed by repeating pre-recorded measurements while injecting exogenous signals into the system. It is shown that these attacks can be detected by injecting random control signal unknown to the attacker, though the feasibility of such random signal injections has not been studied in power grids. Smith described a potent form of attack known as *covert attack*[179] in which the adversary is able to alter the plant behavior by a decoupling structure, while modifying the sensor signals to remain undetected. This requires an omniscient attacker to know the system dynamics and measurement topology and gain complete interception capabilities over all sensor and actuator communications that may

not be feasible in power grids due to the sheer scale of entire system operation. In addition, different works[180–182] have focused on other attack modes on sensor measurements that corrupt the system states. Recently, power researchers have studied a new form of attack known as *topology attack*[183, 184]. This attack targets the network topology data used in various data processing modules in the EMS. Changes in this topology data can result in incorrect generation control with potentially disastrous consequences. The attacker alters the measurements and line status information to make it appear that the corresponding line is not active at the control center via SCADA data. The fundamental mechanism of all these attacks is to corrupt the system states directly (through actuation compromise) or indirectly by state estimation inaccuracy (through sensor data alteration) while concealing intrusion footprints.

As countermeasures, all previous research has targeted some form of secure state estimation designed for specific attacks. Sastry et al.[185] first explored the problem of state estimation in control systems over unreliable communication networks where packet transmission is not trustworthy. The disturbances are however modeled as stochastic distributions that does not capture an intelligent attacker behavior. The fundamental limitations of attack detection and diagnosis are studied by Pasqualetti et al. in [186, 187] for linear systems and specifically for power networks. The authors propose centralized and decentralized filters for attack detection and identification. These filters are computationally expensive and difficult to implement as pointed out in [188]. Bobba et al. demonstrated in [189] that a strategically selected basic set of measurements needs to be protected to thwart least-effort adversarial attacks. This ensures that the attacker will require considerable resources to launch a successful attack. Recently, sparse optimization techniques have evolved as promising tools in secure state estimation [188, 190] where the state estimation is posed as an ℓ_0-norm minimization problem. Though the theoretical and algorithmic aspects are of sound nature, the secure estimation methods are proposed without tackling the attack detection problem. In addition, the mitigation latency of proposed techniques has not

been mentioned. Li et al.[191] recently proposed a methodology for quickest attack detection in a centralized and distributed manner. Firstly, the authors assume that attackers have limited information about network topology and are unable to design undetectable attack vectors and secondly, they assume that such adversaries have limited resources to access the sensor communications. With ever-increasing adversarial capabilities, these assumptions may not hold true for future intelligent attackers. Additionally, the effect of shifting noise statistic on the detection and mitigation accuracies has not been explored properly. The present research aims to propose a comprehensive detection, diagnosis and mitigation mechanism without enforcing any restricting assumptions on attackers.

Mathematical Notations

Here, we define the notations used in this paper. $\Re^n$ defines the n-dimensional real space. $\mathbf{x} \in \Re^n$ indicates that $\mathbf{x}$ is an n-dimensional real vector $[x_1, x_2, \cdots, x_n]^\intercal$. $||\mathbf{x}||_0$, $||\mathbf{x}||_1$ and $||\mathbf{x}||_2$ are the $\mathcal{L}_0$-norm, $\mathcal{L}_1$-norm and $\mathcal{L}_2$-norm of the vector $\mathbf{x}$. Φ defines a null set. A diagonal matrix is indicated by diag$(z_1, z_2, \cdots, z_n)$ where the off-diagonal elements are 0 and the diagonal elements are $z_1, z_2, \cdots, z_n$. The support of a vector $\mathbf{x} \in \Re^n$ is the set of nonzero components of the vector $\mathbf{x}$ and denoted as supp$(x) = \{i \in \{1, \cdots, n\} | x_i \neq 0\}$.

6.3 Problem Description

In this section, we provide the problem definition of detection of security attacks and the requirements of a mitigation scheme to avert catastrophic damage.

6.3.1 System Overview

We first introduce the architecture of a centralized cyberphysical system and then describe how a distributed system such as a power grid works.

i) Centralized System

We describe a block diagram of a cyberphysical system and then show how a power grid is implemented as a distributed system.

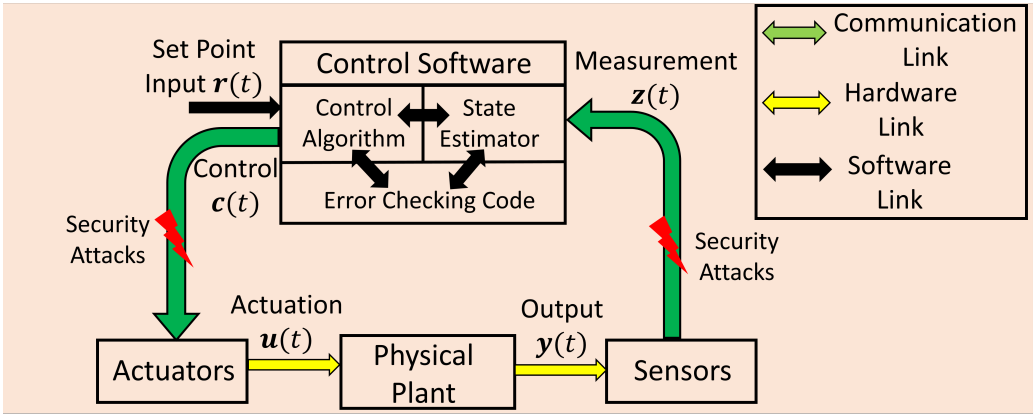


Figure 6.1: Block diagram of a centralized cyberphysical system

In Figure 6.1, the physical plant represents the actual process that needs to be controlled. The dynamics for a linear system is represented by the following state equations:

$$\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + B\mathbf{u}(t) \tag{6.1a}$$

$$\mathbf{y}(t) = H\mathbf{x}(t) \tag{6.1b}$$

where $\mathbf{x}(t) \in \Re^n$ and $\mathbf{u}(t) \in \Re^m$ denote the system states and the plant input, respectively. The plant output is designated as $\mathbf{y}(t) \in \Re^p$. The matrices $A \in \Re^{n\times n}$, $B \in \Re^{n\times m}$ and $H \in \Re^{p\times n}$ completely define and model the linear physical plant in accordance with equations (6.1a) and (6.1b).

The plant is driven by the actuation signal $\mathbf{u}(t)$ over hardware links that connect the plant and the actuators. The output signal $\mathbf{y}(t)$ from the plant is measured through sensors over similar hardware links. The sensors and actuators transmit signals over communication links to the control software. These signal transmissions are vulnerable to external

security attacks from malicious adversaries. The corrupted measurement received by the control software is denoted by $\mathbf{z}(t)$ and modeled as

$$\mathbf{z}(t) = \mathbf{y}(t) + \mathbf{a}_y(t) \tag{6.2}$$

where $\mathbf{a}_y(t)$ is the sensor attack signal. Similarly, the actuator attack signal $\mathbf{a}_x(t)$ corrupts the control signal as

$$\mathbf{u}(t) = \mathbf{c}(t) + \mathbf{a}_x(t) \tag{6.3}$$

The primary reason of separating output signal $\mathbf{y}(t)$ and measurement signal $\mathbf{z}(t)$ as well as actuation signal $\mathbf{u}(t)$ and control signal $\mathbf{c}(t)$ is to explicitly enforce the hardware-software interface present in cyberphysical systems indicating that specific hardware components are isolated from the cyber-layer and cannot be remotely hacked.

Within the control software module, the state estimator estimates the plant states from the measurement $\mathbf{z}(t)$ as

$$\hat{\mathbf{x}}(t) = P(\mathbf{z}(t)) \tag{6.4}$$

where $\hat{\mathbf{x}}(t)$ and $P(.)$ denote the estimated states and the state estimation algorithm, respectively. The estimated state trajectory $\hat{\mathbf{x}}(t) \quad \forall t$ and the desired set point input $\mathbf{r}(t)$ are used by the control algorithm to compute the appropriate control signal as

$$\mathbf{c}(t) = f_c(\hat{\mathbf{x}}(t), \mathbf{r}(t)) \tag{6.5}$$

where $f_c(.)$ denotes the implemented control algorithm. In addition, the control software module also performs other encryption and data integrity functions.

ii) Distributed System

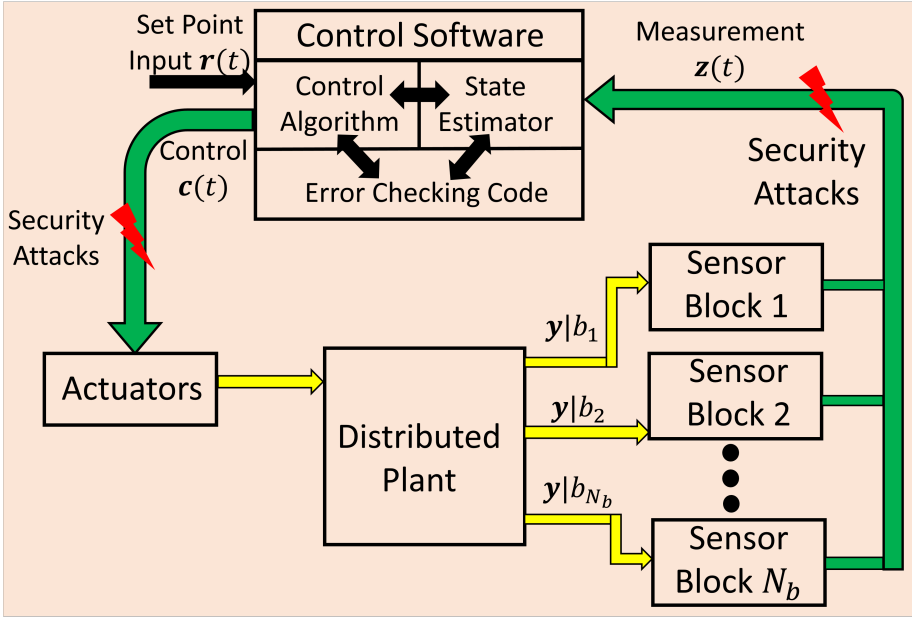


Figure 6.2: Block diagram of a distributed cyberphysical system

Figure 6.2 show the block diagram for a distributed cyberphysical system. Most often, huge cyberphysical systems are distributed over wide geographical areas and the physical plant as well as the sensor nodes are not restricted to a single location. The primary difference between a centralized and a distributed system is that there are N_b sensor blocks measuring the outputs of the distributed plant instead of a single sensor block. Each of the blocks has an independent measurement index set $b_i \subseteq \{1, 2, \cdots, p\} \forall\ i = 1$ to N_b as a subset of the complete measurement set and indicates the outputs sensed by that particular block. For example, a system with 8 measurements and 2 blocks with measurement index sets $\{1, 2, 6, 7\}$ and $\{3, 4, 5, 8\}$ indicates that block 1 senses y_1, y_2, y_6 and y_7 whereas block 2 senses y_3, y_4, y_5 and y_8. The sensed output and transmitted measurement of block

i are represented by $\mathbf{y}|b_i(t)$ and $\mathbf{z}|b_i(t)$ respectively. The central controller forms the complete measurement vector $\mathbf{z}(t)$ from the individual outputs transmitted by each block as $\mathbf{z}(t) = \bigcup_i(\mathbf{z}|b_i(t))$.

6.3.2 Problem Definition

In the absence of security attacks, there is no deviation of the plant performance from the desired trend and the state trajectory follows the intended direction. The estimated state trajectory $\hat{\mathbf{x}}(t)$ tracks the actual plant state trajectory $\mathbf{x}(t)$ closely. The accurate state estimation results in the generation of accurate control signal that is imparted to the plant through the actuators.

In the presence of security attacks, the system states start deviating from the desired trajectory. This occurs due to two primary reasons: i) Firstly, in presence of sensor attacks, the measurements are corrupted and hence the estimated states are different from the actual system states. The control signal is computed based on the erroneous estimated states and hence an incorrect actuation is imparted to the plant leading to deterioration in performance. ii) Secondly, in presence of actuator attacks, the plant is driven by a malicious signal instead of the control signal generated from the controller resulting in incorrect plant states causing eventual system damage. Practical attack strategies typically contain both modes as discussed further in Section 6.5. The detection, diagnosis and recovery problems in presence of security attacks are posed as:

Detection: Develop a methodology that detects any external attacks with low latency.

Diagnosis: Identify the sensor and actuator nodes under attack so that appropriate mitigation schemes can be applied.

Recovery: Generate the true state estimate from the compromised measurements $\mathbf{z}(t)$ in presence of security attacks such that accurate control signal $\mathbf{c}(t)$ can be applied to the plant and isolate the attacked nodes for actuator attacks to restrict the extent of disruption.

6.4 Preliminaries

6.4.1 State Estimation

In the control module of the power grids, the PSSE methods perform state estimation from the received measurements. These state estimates are utilized to compute the optimal generation dispatch by solving an *AC power flow model*[192, 193] where both active and reactive powers are considered. The state variables $\mathbf{x}$ and measurements $\mathbf{z}$ are related through the nonlinear model

$$\mathbf{z} = \mathbf{h}(\mathbf{x}) + \mathbf{n} \tag{6.6}$$

where $\mathbf{h}(\mathbf{x})$ is a nonlinear measurement function of state variable $\mathbf{x}$. The term $\mathbf{n}$ denotes measurement noise with well-modeled probability distribution. In large power systems, performing optimal network flow analysis using *AC power flow model* at each time instant is computationally prohibitive (due to nonlinear equation solving) and impractical in most situations. Typically, there are incessant small load perturbations around a fixed operating point and hence a linearized power flow model known as *DC power flow model*[194] is utilized for state estimation. Employing the *DC power flow model* modifies (6.6) into

$$\mathbf{z} = H\mathbf{x} + \mathbf{n} \tag{6.7}$$

where $H \in \Re^{p \times n}$ is the measurement matrix obtained from the Jacobian of the function $\mathbf{h}(\mathbf{x})$ for a particular operating point. With the measurement noise having a Gaussian distribution with zero mean and known non-zero variances, the *maximum likelihood estimator*[195] is given as

$$\hat{\mathbf{x}} = (H^{\intercal} W H)^{-1} H^{\intercal} W z \tag{6.8}$$

where W is a diagonal matrix with the diagonal elements as reciprocals of noise variances, $W = \text{diag}(\sigma_1^{-2}, \sigma_2^{-2}, \cdots, \sigma_p^{-2})$ with σ_i^2 as the variance of the i-th sensor measurement noise

The error detection is typically performed by a *bad data detector*[194, 196] that checks for any digression of estimated state variables away from their true values. This is computed by the *measurement residual* defined as $\mathbf{e}_r = \mathbf{z} - H\hat{\mathbf{x}}$. This term computes the difference between the observed measurements and the measurements corresponding to the estimated states. The presence of bad measurements is identified by checking if $||\mathbf{e}_r||_2 > \tau$, where τ is a real threshold. The error coverage solely depends on the selection of τ and this is done through statistical measures such as hypothesis test[195]. Bad data detectors are very efficient in detecting outliers and abnormal measurements. In section 6.5, we discuss how attack vectors can be carefully designed to remain undetected by this technique.

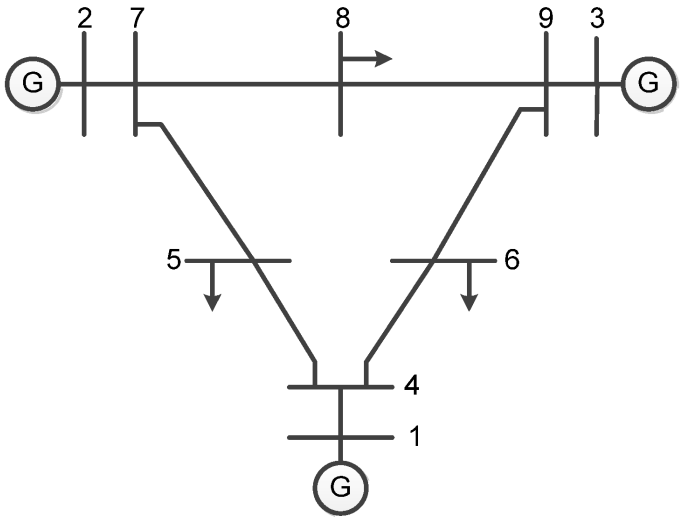


Figure 6.3: IEEE 9-bus system

6.4.2 Test Cases

For this work, we conduct experiments using the IEEE benchmark test systems. The configuration of the systems are extracted using MATPOWER, an open-source MATLAB package for solving power flow problems[197]. As a sample test case, the IEEE 9-bus system is shown in Figure 6.3. 3 generators are connected to buses 1, 2 and 3. Buses 5, 6, and 8 are load buses where power is withdrawn from the network. The other buses are used for routing power through the network. The connections between the buses form the branches.

6.4.3 Example Case

In order to explain the proposed methodologies of this research, a system is considered with 5 states as $\mathbf{x} = [x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5]^\intercal$ and 8 measurements as $\mathbf{y} = [y_1 \quad y_2 \quad y_3 \quad y_4 \quad y_5 \quad y_6 \quad y_7 \quad y_8]^\intercal$. The measurement matrix H is defined in the measurement equation as

$$\mathbf{y} = H\mathbf{x} \tag{6.9}$$

$$= \begin{bmatrix} 1 & 3 & 0 & 0 & 0 \\ -2 & 0 & 0 & 4 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ -1 & 1 & 0 & 1 & 0 \\ 2 & -1 & 0 & 3 & 0 \\ 0 & 4 & -1 & 0 & 2 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & -2 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} \tag{6.10}$$

6.5 Security Attack Models

Prior to defining the security attack models, we assume the following capabilities and restrictions of an attacker:

1. Attacker has knowledge of the measurement matrix H for one or more operating points prior to launching attacks.

2. Attacker does not have knowledge of the power grid dynamics and cannot compute the measurement matrix H by linearizing around any arbitrary operating point.

3. Attacker has sufficient computational resources to generate attack vectors

4. Attacker has the cyber-capability to intercept sensor and actuator communications and inject his own attack vectors into the data transmissions.

5. Attacker doesn't have physical access to the power grid infrastructure and is unable to perform physical compromise (damage to generators/transformers) to the system.

A complete attack vector is modeled as $\mathbf{a}(t) = [\mathbf{a}_x(t), \mathbf{a}_y(t)] \in \Re^{m+p}$ (henceforth, we drop the time designation t unless explicitly needed) where $\mathbf{a}_x$ and $\mathbf{a}_y$ are actuator and sensor attack vectors respectively. The *actuator attack* $[\mathbf{a}_x, \mathbf{0}]^\intercal$ directly affects the system dynamics and the *sensor attack* $[\mathbf{0}, \mathbf{a}_y]^\intercal$ corrupts the measurements. Two different attack sets K_x and K_y are defined for actuator and sensor attacks. An attack set is defined as the index set for which the entries of the attack vector are non-zero over time. For example, if the sensor attack set is $K_y = \{4, p-1\}$, then the attack signal is defined as $\boldsymbol{a}_y = [0, 0, 0, a_4, 0, \cdots, 0, a_{p-1}, 0]$, i.e. only the 4th and $(p-1)$th sensors are compromised. We describe the cardinality of an attack set as $|K|$ that defines the number of nodes under attack. This redefines the diagnosis problem posed before as:

Diagnosis: Identify the sensor and actuator attack sets K_y and K_x upon detection of attacks.

6.5.1 Attack Types

We define two attack types that are particular instances of the general framework introduced: i) stealth attacks where only sensors are affected, and ii) replay attacks where both sensors and actuators are affected. There is no viable attack scheme where only actuators are compromised since without sensor attacks the footprints of actuator attacks are detected in the measurements.

1) Stealth Attack: A stealth attack is a form of sensor attack in which the attacker modifies some or all sensor measurements by gaining access to the communication channels. This is modeled by the attack vector $\mathbf{a} = [\mathbf{0}, \mathbf{a}_y]^\intercal$ with attack sets $K_x = \Phi$ and $K_y \subseteq \{1, 2, \cdots, p\}$ (only the measurements are compromised). The sensor nodes under attack are identified by the non-zero indexes of the attack set. Stealth attacks have two important features. First, they can be designed with the knowledge of the measurement topology only without any information about the system dynamics and second, they are undetectable by *bad data detector*[194, 196] described previously.

2) Replay Attack: A replay attack is a combined sensor and actuator attack scheme characterized by three primary actions of the attacker: i) the attacker records the system outputs and inputs corresponding to a nominal state of operation, ii) next, the attacker waits to observe similar actuator inputs and corrupts them to disrupt the system dynamics, and iii) finally, the sensor measurements are altered to replicate previously recorded measurements corresponding to nominal operating conditions. Replay attacks are modeled as $\mathbf{a} = [\mathbf{a}_x, -H(\mathbf{x}_a) + H(\mathbf{x})]^\intercal$ where $\mathbf{x}_a$ and $\mathbf{x}$ are compromised system states under attack and nominal states without attack, respectively. Replay attacks can be carried out without any knowledge of system dynamics, similar to stealth attacks, but this requires the attacker to gain access to actuators in addition to all sensor nodes.

The security attacks are depicted as block diagram models in Figure 6.4. We next discuss the generation strategies of these attacks.

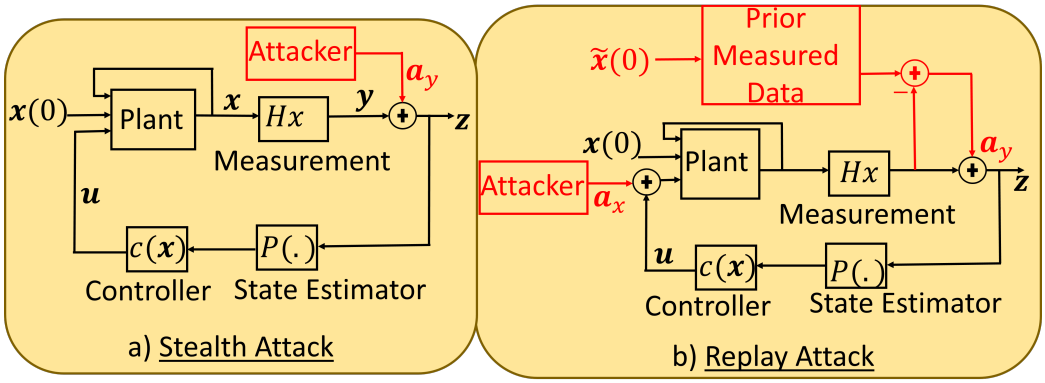


Figure 6.4: Schematic Diagram of the different attack models: Blocks in red denote attacker actions

6.5.2 Attack Generation Strategies

1) Stealth Attack: The basic principle of stealth attack, also known as *false data injection attack*[172], is to inject an attack vector that lies in the column space of the measurement matrix H, thus ensuring that the additional error stays in the null space of the residual based error detection scheme and is undetectable.

Let $\mathbf{z}$ be the attack-free measurement with the correct estimated state as $\hat{\mathbf{x}}$. In absence of attacks, the residual $\mathbf{e}_r = ||\mathbf{z} - H\hat{\mathbf{x}}||_2 \leq \tau$, where τ is the detection threshold. Suppose, with the injection of an attack $\mathbf{a}$, the corrupted measurement is $\mathbf{z}_a = \mathbf{z} + \mathbf{a}$ that results in the erroneous state estimate $\hat{\mathbf{x}}_a = \hat{\mathbf{x}} + \mathbf{c}$ where $\mathbf{c}$ reflects the state estimation error. It can be shown[172] that if the attack vector $\mathbf{a}$ is generated to lie in the column space of H as $\mathbf{a} = H\mathbf{c}$ with $\mathbf{c}$ as any arbitrary non-zero vector, it is undetectable by residual based detectors. This can be proved in computing the measurement residual as

$$\begin{aligned} ||\mathbf{z}_a - H\hat{\mathbf{x}}_a||_2 &= ||\mathbf{z} + \mathbf{a} - H(\hat{\mathbf{x}} + \mathbf{c})||_2 \\ &= ||(\mathbf{z} - H\hat{\mathbf{x}}) + (\mathbf{a} - H\mathbf{c})||_2 \\ &= ||\mathbf{z} - H\hat{\mathbf{x}}||_2 \leq \tau \end{aligned} \tag{6.11}$$

Hence, the corrupted measurement $\mathbf{z}_a$ will be undetectable through the bad data detector. From practical perspective, the attacker rarely has access to all sensor nodes and can only compromise a subset of the sensor nodes defined by the attack set K_y. The generation of the stealth attacks has been described in details in [172] and briefly summarized in Algorithm 6 described in Appendix C.

2) Replay Attack - For replay attack, an attacker collects measurement data before launching an attack. Suppose the attacker has recorded measurement data for T_r time samples from a previous arbitrary time instant $t = t_p$ along with the historical load demand data. Let the pre-recorded measurement data be $\mathbf{z}_p(t)$ for $t = t_p, t_p + 1, \cdots, t_p + T_r - 1$. A replay attack is launched by the attacker when the present load pattern resembles the pre-recorded load pattern such that the attack injection does not trigger any watchdog mechanism designed to identify abnormal power flow variations. At the moment of attack injection, say at time $t = t_0$, the attacker replaces the actual sensor measurement data with the pre-recorded data as $\mathbf{z}(t = t_0) = \mathbf{z}_p(t = tp)$. Simultaneously, the attacker injects malicious signals in the actuation data and these can be chosen as any arbitrary signal as $\mathbf{a}_x(t = t_0) = \mathbf{f}(t)$ where $\mathbf{f}(t)$ can be any chosen vector function. The substitution of the measurements with pre-recorded sensor data ensures that the state estimation techniques don't detect any irregularity in system operation while the disruptive effect of actuator attack is underway. The attack injection continues with the adversary continually replacing the sensor data with pre-recorded measurements while corrupting actuation signals.

6.6 Proposed Attack Detection Methodology

The primary objective of an attack detection scheme is to generate a non-zero error signal in presence of external security intrusions. First, we propose a methodology for detection of security attacks in a centralized system of Figure 6.1 and then present a hierarchical architecture for distributed systems of Figure 6.2.

6.6.1 Centralized Detection

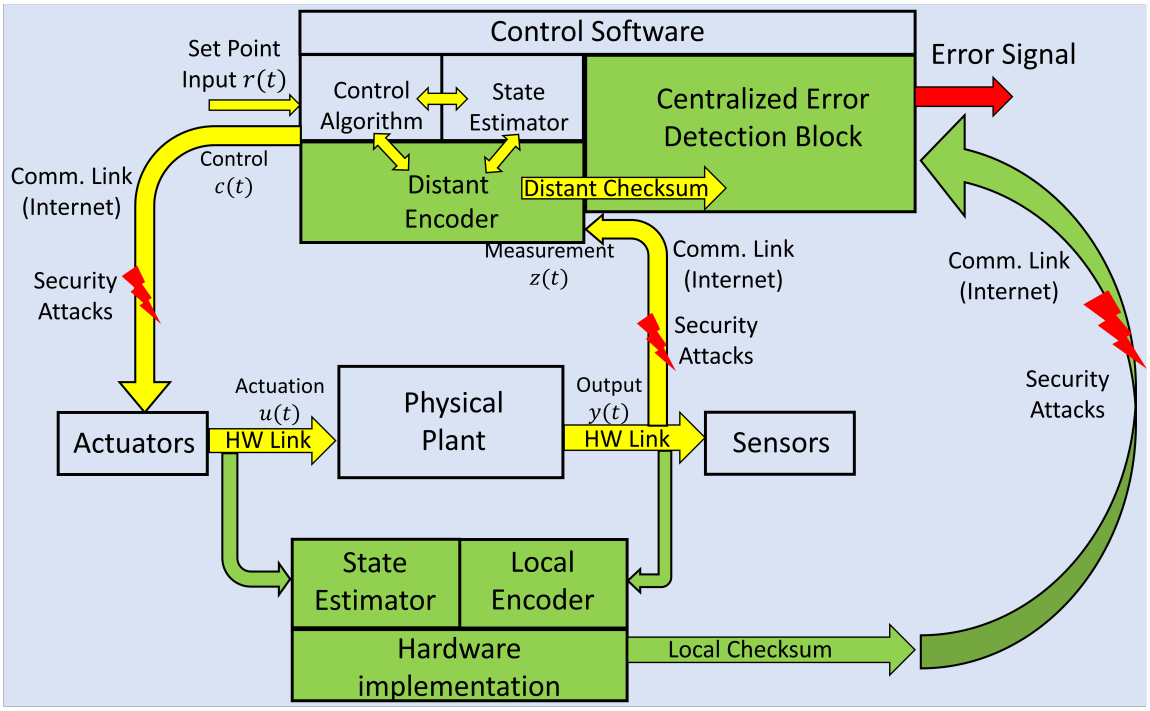


Figure 6.5: Proposed error detection methodology for security attacks

Figure 6.5 demonstrates the block diagram of proposed error detection methodology for security attacks. The key feature of this scheme is to perform a state estimation from the measurement $\mathbf{y}(t)$ in close proximity to the actual physical system and generate an encoded check state that is transmitted back to the supervisory control module. This local implementation of state estimator and encoder is designed such that the inputs for this module can be tapped from the actual sensors. The local block is implemented on a secure hardware platform or a software module in a highly secure environment to ensure that external security attacks cannot affect the operation of this module. The local state estimation is performed as

$$\hat{\mathbf{x}}_l(t) = P_l(\mathbf{y}(t)) \tag{6.12}$$

where $\hat{\mathbf{x}}_l(t)$ is the local estimated state and $P_l(.)$ is the local implementation of the state estimator. For example, in the DC state estimation of power grids, $P_l(.)$ implements the maximum likelihood estimator in (6.8). In a generic cyber-physical system, $P_l(.)$ implements Kalman filter or similar state estimators. A local encoder computes a check state from the local estimated states $\hat{\mathbf{x}}_l(t)$ as

$$e_l(t) = f_l(\hat{\mathbf{x}}_l(t)) \tag{6.13}$$

where $e_l(t)$ is the local check state and $f_l(.)$ is the local encoding function. $e_l(t)$ is a single encoded check state generated from the n estimated states. In addition to the sensor signals, this local check state $e_l(t)$ is also transmitted to the control software over communication links. The control software estimates the states from $\mathbf{z}(t)$ in accordance with (6.4) as $\hat{\mathbf{x}}(t)$ that are redefined here as $\hat{\mathbf{x}}_d(t)$ to indicate the distant estimated states (in contrast with local estimated states $\hat{\mathbf{x}}_l(t)$). The central controller computes a separate encoded state from $\hat{\mathbf{x}}_d(t)$, known as the distant check state as,

$$e_d(t) = f_d(\hat{\mathbf{x}}_d(t)) \tag{6.14}$$

where $e_d(t)$ is the distant check state and $f_d(.)$ is the implemented encoding. The centralized error detection block compares the received local check state $e_l(t)$ and the generated distant check state $e_d(t)$ to form the error signal $e(t)$ defined as

$$e(t) \triangleq e_d(t) - e_l(t) \tag{6.15}$$

Under nominal operating conditions without any external attacks, the output signal $\mathbf{y}(t)$

is the same as the measured signal $\mathbf{z}(t)$. If noise sources have well-modeled probability distributions, the local estimated state $\hat{\mathbf{x}}_l(t)$ and the distant estimated state $\hat{\mathbf{x}}_d(t)$ are also equal. Enforcing proper selection of $f_l(.)$ and $f_d(.)$ leads to the same check states generated both locally and distantly. Hence the error signal $e(t)$ is zero. From a practical perspective, the nominal error signal lies within a small threshold due to noise. Under security attacks, the local and distant estimated states are different since these are computed from unattacked and attacked measurement signals. Hence, the local check state $e_l(t)$ and the distant check state $e_d(t)$ are dissimilar due to difference in estimated states $\hat{\mathbf{x}}_d(t)$ and $\hat{\mathbf{x}}_l(t)$, resulting in non-zero error signal $e(t)$.

The simplest choice of encoding function $f_l(.)$ in (6.13) is a row vector $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \cdots, \alpha_n]$ with linear weights $\alpha_i \in \Re$. Similarly, the distant encoder also implements the function $f_d(.)$ with a separate row vector $\boldsymbol{\beta} = [\beta_1, \beta_2, \cdots, \beta_n]$ with $\beta_i \in \Re$. With $\boldsymbol{\alpha} = \boldsymbol{\beta}$, the error signal $e(t)$ in (6.22) is zero in the absence of any attacks, since both the local and distant estimated states are also same.

Example: For the system mentioned in 6.4.3, the local state estimator predicts the state vector $\hat{\mathbf{x}}_l$ from the measurements $\mathbf{y}$ as

$$\hat{\mathbf{x}}_l = [\hat{x}_{1(l)} \quad \hat{x}_{2(l)} \quad \hat{x}_{3(l)} \quad \hat{x}_{4(l)} \quad \hat{x}_{5(l)}]^\intercal \tag{6.16}$$

$$= P_l(\mathbf{y}) = (H^\intercal W_l H)^{-1} H^\intercal W_l \mathbf{y} \tag{6.17}$$

where W_l is the noise covariance matrix of $\mathbf{y}(t)$. The local check state $e_l(t)$ is computed using the coding vector $\boldsymbol{\alpha} = [\alpha_1 \quad \alpha_2 \quad \alpha_3 \quad \alpha_4 \quad \alpha_5]$ as

$$e_l = \alpha_1 \hat{x}_{1(l)} + \alpha_2 \hat{x}_{2(l)} + \alpha_3 \hat{x}_{3(l)} + \alpha_4 \hat{x}_{4(l)} + \alpha_5 \hat{x}_{5(l)} \tag{6.18}$$

In presence of sensor attacks, the central controller receives the corrupt signal $\mathbf{z}(t) = \mathbf{y}(t) + \mathbf{a}_y(t)$. The predicted state by the estimator in the central controller is given as,

$$\hat{\mathbf{x}}_d = [\hat{x}_{1(d)} \quad \hat{x}_{2(d)} \quad \hat{x}_{3(d)} \quad \hat{x}_{4(d)} \quad \hat{x}_{5(d)}]^\intercal \tag{6.19}$$

$$= P_d(\mathbf{z}) = (H^\intercal W_d H)^{-1} H^\intercal W_d \mathbf{z} \tag{6.20}$$

where W_d is the noise covariance matrix of $\mathbf{z}(t)$.

The distant check state $e_d(t)$ computed by the central controller using the linear coding vector $\boldsymbol{\beta} = [\beta_1 \quad \beta_2 \quad \beta_3 \quad \beta_4 \quad \beta_5]$ is given as

$$e_d = \beta_1 \hat{x}_{1(d)} + \beta_2 \hat{x}_{2(d)} + \beta_3 \hat{x}_{3(d)} + \beta_4 \hat{x}_{4(d)} + \beta_5 \hat{x}_{5(d)} \tag{6.21}$$

The central error detection module generates the error signal $e(t)$ as

$$e(t) \triangleq e_d(t) - e_l(t) \tag{6.22}$$

Without any external attacks, $\mathbf{y}(t) = \mathbf{z}(t)$ and hence $\hat{\mathbf{x}}_l(t) = \hat{\mathbf{x}}_d(t)$. If $\boldsymbol{\alpha} = \boldsymbol{\beta}$, then the two check states $e_l(t)$ and $e_d(t)$ are equal and hence $e = 0$. If $\mathbf{y}(t) \neq \mathbf{z}(t)$, then the two check states are unequal and $e(t) \neq 0$.

6.6.2 Distributed Detection

For a distributed cyberphysical system, the local state estimator and encoder in Figure 6.5 needs to be located in close proximity to the individual sensor measurement blocks of Figure 6.2. In this work, we refer "*area*" as a region small enough such that a single control center is sufficient to manage the operation. In systems of enormous size, such as

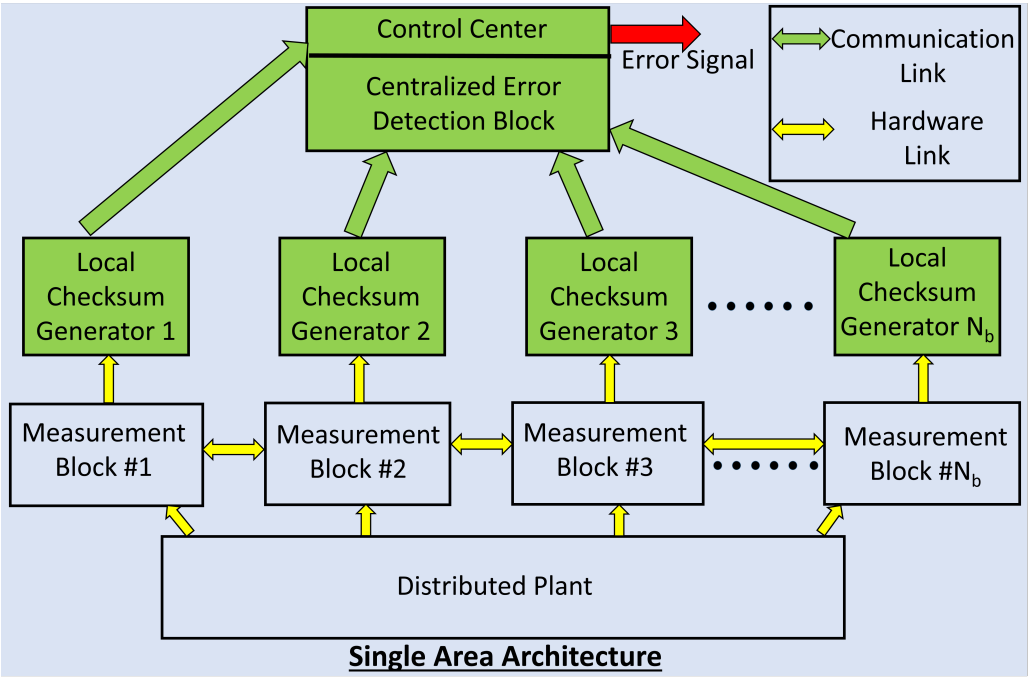


Figure 6.6: Distributed Architecture for detection of security attacks in a single area

the Eastern Interconnection Grid spanning several states in USA, the entire cyber-physical system is divided into different areas, each with its own control center. The areas exchange information between neighboring areas and control the entire operation in a distributed manner. First, we propose error detection in a single area architecture as illustrated in Figure 6.6 and next we extend the concept to a multi-area setting.

In figure 6.6, the sensors access the distributed plant in different blocks separated from each other. Each of the N_b blocks has two sets associated with it: i) measurement index set b_i previously described in Section 6.3.1 and ii) state index set s_i $\forall i = 1$ to N_b. The measurement index set b_i determines the outputs that are sensed in each block and the state index set determines the subset of the system states $\mathbf{x}$ that can be estimated in each block. Due to the geographical spread, the measurement matrix H is typically sparse and individual measurements of the output vector $\mathbf{y}$ depend on a small subset of the state vector $\mathbf{x}$. The number of sensors grouped in each block depends on the actual physical plant installation and the sensor locations.

For $i = 1$ to N_b, the i-th local block takes as input, the measurements of the i-th block $\mathbf{y}|b_i$. From the measurements, the ith local block estimates a subset of the system states as $\hat{\mathbf{x}}_{l|s_i}$ and the encoder computes the local check state as,

$$e_{l(i)} = f_{l(i)}(\hat{\mathbf{x}}_{l|s_i}) \tag{6.23}$$

where $e_{l(i)}$ is the check state generated by the i-th local block, $f_{l(i)}$ is the encoding mechanism of the i-th local block and $\hat{\mathbf{x}}_{l|s_i}$ are the local estimation of states indexed by s_i. All the local blocks transmit their respective check states $e_{l(i)}$s to the central control center. The central controller has complete knowledge of the measurement index sets b_is and state index sets s_is for all the blocks. After receiving all the measurements $\mathbf{y}|b_i \; \forall i = 1$ to N_b, the controller estimates the entire system state $\hat{\mathbf{x}}_d$ and computes the distant check state in the usual fashion as $e_d = f_d(\hat{\mathbf{x}}_d)$. The error signal is finally computed as,

$$e(t) = e_d(t) - f_e(e_{l(1)}(t), e_{l(2)}(t), \cdots, e_{l(N_b)}(t)) \tag{6.24}$$

where $f_e(.)$ is an appropriate transformation that determines the weights assigned to each local check state $e_{l(i)}$s.

The functions $f_{l(i)}(.) \; \forall i = 1$ to N_b, $f_d(.)$ and $f_e(.)$ are carefully chosen such that under nominal conditions without attack, the error signal defined in (6.24) lies close to zero. The distributed detection methodology is presented in Algorithm 3 where the sparsity of the measurement matrix H is exploited to determine the state index sets thus indicating the states that can be estimated in each block. The execution of the algorithm is provided in a following example.

For the single-area distributed detection, the state index set $s_i \; \forall i = 1$ to N_b is defined as an n-length vector $\mathbf{s}_i = [s_{i1}, s_{i2}, \cdots, s_{in}]$ where the j-th entry s_{ij} is 1 if state x_j can be

Algorithm 3 Distributed Detection

procedure DIST_DETECT
  $N_b \leftarrow$ number of blocks
  for do $i = 1$ to N_b
    $b_i \leftarrow$ measurement set of block i
    $f_{l(i)} \leftarrow$ local encoding of block i
    $H_i \leftarrow$ rows of H indexed by b_i
    $H_i \leftarrow$ zero columns removed from H_i
    $s_i \leftarrow$ column indexes of H_i from H's indexing
    $\hat{\mathbf{x}}_{l|s_i} \leftarrow$ local state estimate indexed by s_i
    $e_{l(i)} = f_{l(i)}(\hat{\mathbf{x}}_{l|s_i})$
  $f_d \leftarrow$ distant encoding
  $\hat{\mathbf{x}}_d \leftarrow$ distant state estimate
  $e_d = f_d(\hat{\mathbf{x}}_d)$
  $e = e_d - f_e(e_{l(1)}, e_{l(2)}, \cdots, e_{l(N_b)})$

estimated in the i-th block or 0 otherwise. The local encoders $f_{l(i)}$ in (6.23) are defined by separate row vectors $\boldsymbol{\kappa}_1, \boldsymbol{\kappa}_2, \cdots, \boldsymbol{\kappa}_{N_b}$ with $\boldsymbol{\kappa}_i \in \Re^n \; \forall i = 1$ to N_b. Let the function $f_e(.)$ be also chosen as a linear weighting vector $\mathbf{w} = [w_1, w_2, \cdots, w_{N_b}]$ with $w_i \in \Re$ as the weight of each local check state. Then the function $f_d(.)$ is given as

$$f_d(\hat{\mathbf{x}}_d) = \sum_{j=1}^{N_b} w_j.(\boldsymbol{\kappa}_j(*)\mathbf{s}_j).\hat{\mathbf{x}}_d \tag{6.25}$$

where $(*)$ is the Schur product or entry-wise product between two vectors. The block weighting vector $\mathbf{w}$ is important to accommodate altered noise statistics. If the noise statistics of the ith block changes from the modeled distribution, the noise power of the estimated states indexed by s_i increases. By reducing the i-th block weight w_i, the noise power in the error signal can be decreased, thus making it difficult for an attacker to conceal an attack signature under evolving noise statistics.

Example: Let the 8 measurements in the system described in Section 6.4.3 be divided into 2 measurement blocks B_1 and B_2. Each of the two blocks is indexed by the measure-

ment sets $\mathbf{b}_1$ and $\mathbf{b}_2$ as $\mathbf{b}_1 = \{1, 2, 3, 4, 5\}$ and $\mathbf{b}_2 = \{6, 7, 8\}$. This implies that B_1 senses y_1, y_2, y_3, y_4 and y_5 whereas B_2 senses y_6, y_7 and y_8. To find out the state index set of B_1 and B_2, DIST_DETECT is used. Let H_1 be the matrix with rows of H indexed by b_1. Thus H_1 is given as

$$H_1 = \begin{bmatrix} 1 & 3 & 0 & 0 & 0 \\ -2 & 0 & 0 & 4 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ -1 & 1 & 0 & 1 & 0 \\ 2 & -1 & 0 & 3 & 0 \end{bmatrix} \tag{6.26}$$

Now, after removing zero columns of H_1, we get

$$H_1 = \begin{bmatrix} 1 & 3 & 0 \\ -2 & 0 & 4 \\ 0 & 1 & 1 \\ -1 & 1 & 1 \\ 2 & -1 & 3 \end{bmatrix} \tag{6.27}$$

$\mathbf{s}_1$ is the index set of H_1's columns according to the original column indexing of H. Thus $\mathbf{s}_1 = [1 \quad 1 \quad 0 \quad 1 \quad 0]$ as the 1st, 2nd and 4th columns of H are contained in H_1. Thus the estimated states in B_1 are $\hat{\mathbf{x}}_{l|s_1} = [\hat{x}_{1(l)}, \hat{x}_{2(l)}, \hat{x}_{4(l)}]^\intercal$. Similarly, $\mathbf{s}_2$ is computed as $\mathbf{s}_2 = [0 \quad 1 \quad 1 \quad 0 \quad 1]$ and $\hat{\mathbf{x}}_{l|s_2} = [\hat{x}_{2(l)}, \hat{x}_{3(l)}, \hat{x}_{5(l)}]^\intercal$. This implies that the local estimator of block B_1 predicts x_1, x_2 and x_4 whereas the estimator of block B_2 predicts x_2, x_3 and x_5. Now the local check state $e_{l(1)}$ is computed with the linear coding vector $\boldsymbol{\kappa}_1 = [\kappa_{11} \quad \kappa_{12} \quad \kappa_{13} \quad \kappa_{14} \quad \kappa_{15}]$ as

$$e_{l(1)} = [\boldsymbol{\kappa}_1(*)\mathbf{s}_1].\hat{\mathbf{x}}_l \tag{6.28}$$

$$= \kappa_{11}\hat{x}_{1(l)} + \kappa_{12}\hat{x}_{2(l)} + \kappa_{14}\hat{x}_{4(l)} \tag{6.29}$$

where $(*)$ is the Schur product or entry-wise product between two vectors. Similarly the check state of block B_2 is expressed as $e_{l(2)} = [\boldsymbol{\kappa}_2(*)\mathbf{s}_2].\hat{\mathbf{x}}_l$ where $\boldsymbol{\kappa}_2 = [\kappa_{21} \quad \kappa_{22} \quad \kappa_{23} \quad \kappa_{24} \quad \kappa_{25}]$ is the coding vector of block 2. The two check states $e_{l(1)}$ and $e_{l(2)}$ are transmitted back to the central controller which has knowledge of the complete measurement matrix H and estimates $\hat{\mathbf{x}}_d$ from $\mathbf{z}$. The central controller computes the check state $e_d = f_d(\hat{\mathbf{x}}_d)$. Let the weight of the two check states is determined by a linear weighting vector $\mathbf{w} = [w_1 \quad w_2]$. Then the final error is defined as,

$$e = e_d - (w_1 e_{l(1)} + w_2 e_{l(2)}) \tag{6.30}$$

For the attack-free error signal to be zero, the function $f_d(.)$ is chosen as

$$f_d(\hat{\mathbf{x}}_d) = \sum_{j=1}^{N_b} w_j.(\boldsymbol{\kappa}_j(*)\mathbf{s}_j).\hat{\mathbf{x}}_d \tag{6.31}$$

For multi-area architecture, the error detection scheme operates in a distributed fashion over multiple control centers as illustrated in Figure 6.7. Let, the entire cyber-physical system be divided into N_a areas with each area monitored by its own control center. The control center in each area implements error detection scheme as shown in Figure 6.6. For $i = 1$ to N_a, let the measurement index set and state index set be defined as B_i and S_i, where B_i and S_i determine the measurements and estimated states from the i-th area. The i-th control center receives the estimated states from all other areas to compute the

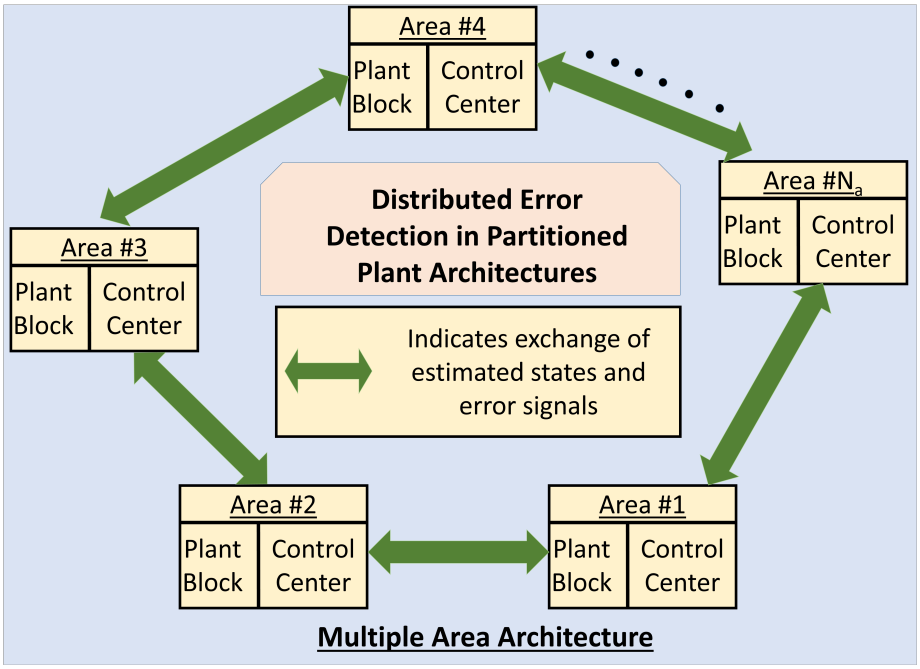


Figure 6.7: Distributed Architecture over multiple areas where each area has its own control center. The different areas share state estimation and error signals among each other.

appropriate actuation signals in the i-th area. The i-th area observes the measurements indexed by set B_i to compute state values indexed by set S_i. From these state estimates, the i-th control center computes the single area error signal $E_i(t)$ defined by equation (6.24). In addition to receiving state estimates from adjacent areas, the i-th control center also receives the error signals from all other areas. If the ith area shares a non-zero error signal, the other areas stop trusting the received state estimates from that area and take appropriate preventive measures.

6.6.3 Resilience of Proposed Scheme

The transmission of local check states over vulnerable communication links makes it possible for an attacker to observe the check signals and attempt to reverse-engineer the encoding scheme so that he can exactly compensate the effect of an injected attack, thus rendering the detection futile. For example, in a stealth attack described in Section 6.5.2, the attacker injects the attack vector such that the corruption in the measurement falls in the null space

of the residual error signal. In the proposed methodology, the two estimated states $\hat{\mathbf{x}}_l(t)$ and $\hat{\mathbf{x}}_d(t)$ are unequal and hence the check states $e_l(t)$ and $e_d(t)$ do not match up, thus triggering the attack detection. If the attacker figures out the change in local check state due to the injected attack, $e_l(t)$ can be adjusted by that exact amount such that the central controller generates an error signal $e(t) = 0$, thus masking the security attack. Injection of an arbitrary signal in the check state generates a non-zero error signal as $e_l(t)$ and $e_d(t)$ are unequal, thus failing to conceal the injected attack. The execution of an adversarial counterattack is difficult due to two primary reasons:

1. It is noteworthy that the check state is a linear encoding of the system states and not the measurements. If the attacker had knowledge of the exact system states, reverse-engineering the linear encoding would be easy for the attacker. As mentioned in Section 6.5, the attacker does not have knowledge of the power grid dynamics and the load changes in the different load buses of the network. Without this information, it is impossible for the attacker to know the exact system states at all times and compromise the local check state transmission.

2. From the attacker's perspective, the brute-force approach requires a black-box modeling with the measurements as inputs and the check state as output. In the absence of knowledge about load changes, it is practically impossible to generate a reverse-engineered model as demonstrated next.

We conduct experiments where we use 2 different tools - a regression method called MARS (Multivariate Adaptive Regression Splines) and a Levenberg-Marquardt based artificial neural network (ANN) to build a model from the measurements to the local checksum signal. The model accuracy of any machine learning tool depends on the diversity of training data.

In figure 6.8, we illustrate the accuracy of the trained MARS model for the 9-bus system shown in Figure 6.3. The model has been built on measurement data for single load changes

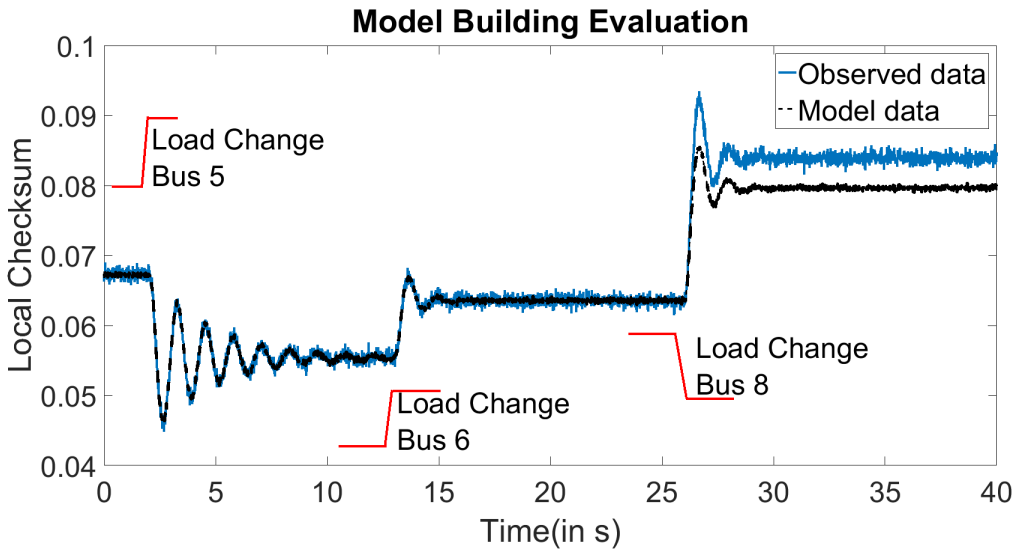


Figure 6.8: MARS model accuracy for single load changes

on bus 5 and 6. It is seen that the trained model accurately predicts the checksum signal for different load changes for those buses but fails to model the same for a load change in bus 8. In order to successfully compromise the detection scheme, the attacker needs to build a model for all combinations of load changes, including simultaneous load perturbations across multiple buses. This is infeasible due to two reasons: firstly, the total number of load perturbations increases exponentially with the increase in number of buses due to the myriad possible combinations and secondly, the different load demands don't occur serially over time for the attacker to collect all the training data required for successful model creation. Even if it can be argued that an attacker has snooped on the transmissions for an elongated period of time to gather a historical set of training data, it becomes prohibitively difficult to build a model due to the sheer amount of data needed. For example, in the 1951-bus system there are 1010 load buses and the number of possible multi-load changes is in the order of 10^{302}. Figure 6.9 illustrates the time needed for successfully building a model using the two different tools.

From Figure 6.9, it is seen that it takes a long time for successful model building in systems with high number of buses ($\approx$ 20 days and $\approx$ 16 hours for 1951-bus and 300-bus networks respectively). We divided up the total data into disparate sets to enable handling

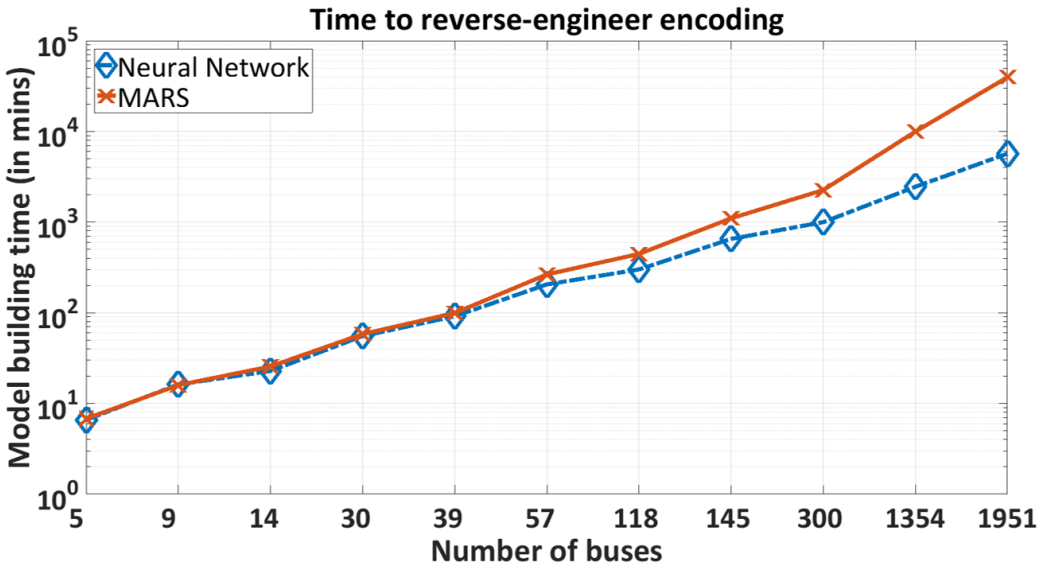


Figure 6.9: Execution time for model building with neural network and MARS

of extremely large data sets during training.

Hence, we propose that the model building attack can be thwarted by *periodically altering both the local and remote coding vectors* such that an attacker can never collect all the data for one particular encoding function. The period of changing the encoding function parameters depends on the scale of the network.

6.6.4 Effects of Noise

All real cyber-physical systems operate in noisy environments and it is imperative to study the effects of noise on the efficacy of proposed methodologies. If the system is too noisy or the stochastic distribution parameters are not well modeled, the error signal $e(t)$ from the detection module is noisy. Under such circumstances, it is possible for an intelligent attacker to inject attacks in a way such that the intrusion footprints stay below the noise floor of the error signal resulting in a detection failure. To study the effects of noise on the detection accuracy, we define two different noise sources - i) *measurement noise* - this refers to the observational noise inherently present in the sensing equipments and ii) *transmission noise* - this refers to the noise added during transmission through wired/wireless links. The

measurement noise $\mathbf{n}$ in (6.7) is redefined as $\mathbf{n}_m(t)$ and the transmission noise is introduced as $\mathbf{n}_t(t)$. The output signal $\mathbf{y}(t)$ is corrupted by the measurement noise $\mathbf{n}_m(t)$ only while the measurement signal $\mathbf{z}(t)$ is additionally corrupted by the transmission noise $\mathbf{n}_t(t)$. Now, we make the following proposition:

Proposition I: *The detection accuracy depends only on the transmission noise* $\mathbf{n}_t(t)$ *and is not affected by the measurement noise* $\mathbf{n}_m(t)$.

Proof: For an arbitrary signal $s(t)$, the power P_s is given as $P_s = \frac{1}{T}\int_0^T s^2(t)dt$ over a time interval of T. The noise power of a zero-mean Gaussian distributed noise $n(t)$ with variance σ_n^2 is $P_n = \sigma_n^2$. Hence, the SNR is defined as SNR $= 10\log_{10}\frac{P_s}{P_n}$. For unattacked systems, let the noisy output signal that is transmitted be $\tilde{\mathbf{y}}(t) = \mathbf{y}(t) + \mathbf{n}_m(t)$. Similarly, the received measurement signal without transmission noise is $\mathbf{z}(t) = \tilde{\mathbf{y}}(t)$. Both the local checksum $e_l(t)$ and the remote checksum $e_d(t)$ are generated from the same noisy data $\tilde{\mathbf{y}}(t)$, thus producing a final error signal of $e(t) = 0$. Operating under attacks, the received measurement signal is $\mathbf{z}(t) = \tilde{\mathbf{y}}(t) + \mathbf{a}_y(t)$. This generates non-zero error signal $e(t)$ with the signal power P_s proportional to the mismatch between local and distant checksum codes due to the attack as $P_s \propto \mathbf{a}_y$ and with noise power $P_n = 0$.

In the presence of transmission noise, the received measurement of an unattacked system is $\tilde{\mathbf{z}}(t) = \tilde{\mathbf{y}}(t) + \mathbf{n}_t(t)$. Hence, the local and distant checksum codes do not exactly match resulting in non-zero error signal $e(t)$ with $P_n \sim \sigma_{n_t}^2$. For attacked systems, the error signal $e(t)$ is non-zero due to i) the injected attack and ii) transmission noise resulting in $P_s \propto \mathbf{a}_y$ and $P_n \sim \sigma_{n_t}^2$. Thus the SNR of the error signal is defined as

$$\begin{aligned} \text{SNR} &= 10\log_{10}\frac{P_s}{P_n} \\ &\propto \frac{\mathbf{a}_y(t)}{\mathbf{n}_t(t)} \end{aligned} \tag{6.32}$$

(6.32) clearly demonstrates that an attacker's capability to camouflage his attack de-

pends on the SNR of the error signal that, in turn, depends on the transmission noise statistic and not affected by the measurement noise $n_m(t)$. From an intuitive perspective, the measurement noise variances are utilized in both the local and distant state estimate. The error signal, being constructed as difference between the local and distant state encodings, cancels out the effect of measurement noise and depends solely on the transmission noise. In Section 6.8.4, we illustrate the effects of transmission noise statistic on the error detection accuracy.

6.7 Proposed Diagnosis and Mitigation Methodology

As described in Section 6.3.2, the objective of the diagnosis algorithm is to identify the sensor and/or actuator attack sets that determines the compromised sensor and/or actuator nodes. The recovery mechanism ignores the measurements from the attacked sensor nodes while performing state estimation in the central controller. Preventive measures are adopted to thwart the attack process and isolate the compromised sensor and/or actuator nodes.

6.7.1 Baseline Scheme

We first present the state-of-the-art diagnosis and correction algorithm[188] as the baseline scheme to compare our proposed technique. Though the system dynamics is described by the continuous state space equation in (6.1), the control center operates on sampled digital measurements. To present the diagnosis and the correction scheme, we first form the discrete form[198] of the linear dynamical equations provided in (6.1) as

$$\mathbf{x}[t+1] = A_d\mathbf{x}[t] + B_d\mathbf{u}[t] \tag{6.33a}$$

$$\mathbf{y}[t] = H\mathbf{x}[t] \tag{6.33b}$$

where $A_d = e^{AT_s}$ and $B_d = \left(\int_{\tau=0}^{T_s} e^{A\tau} d\tau \right) B$ are the discretized forms of the system matrices with T_s as the sampling time.

For ease of explanation, $\mathbf{u}(t)$ is dropped from the initial analysis and is included in a detailed discussion later. Under this assumption, the measurement equation is analogous to the system

$$\mathbf{y}[t] = H A_d^t \mathbf{x}[0] \tag{6.34}$$

with $\mathbf{x}[0]$ as the initial state. The following analysis is valid for the affine system (6.33a). Considering only sensor attacks, the core objective of the diagnosis problem is to observe the measurements $\mathbf{z}[t] = \mathbf{y}[t] + \mathbf{a}_y[t]$ and determine the attack set K_y that provides the indexes of the compromised sensors. The diagnosis algorithm is triggered when the error detection modules discussed in Section 6.6 identifies the presence of an external attack. This instant is chosen as time $t = 0$ for the diagnosis and correction algorithm. The objective of the recovery scheme is to reconstruct the correct state estimates utilizing the knowledge of diagnosed sensor nodes, last known attack-free state and the system model described in (6.33a).

The diagnosis and mitigation is performed by an optimization algorithm which searches for the smallest attack set $K_y \subseteq \{1, 2, \cdots, p\}$ (where p is the total number of measurements) that explains the received measurements over a chosen time horizon of T_h steps. The algorithm observes the measurements $\mathbf{z}[0], \mathbf{z}[1], \cdots, \mathbf{z}[T_h - 1]$ and performs the following optimization problem

$$\begin{aligned} \underset{\hat{\mathbf{x}} \in \Re^n, \hat{K}_y \subseteq \{1, \cdots, p\}}{\text{minimize}} \quad & |\hat{K}_y| \\ \text{subject to} \quad & \text{supp}\left(\mathbf{z}[t] - H A_d^t \hat{\mathbf{x}}[0] \right) \subseteq \hat{K}_y \\ & \text{for} \quad t \in \{0, 1, \cdots, T_h - 1\}. \end{aligned} \tag{6.35}$$

Equation (6.35) shows that the minimization of attack set cardinality is an ℓ_0-norm optimization problem and it has been shown in [188, 199, 200] that such an optimization is NP-hard. However, replacing the ℓ_0-norm optimization by ℓ_1-norm optimization transforms the problem into a convex problem that can be solved efficiently [200]. Hence, the baseline detection and correction algorithm is given in Algorithm 4.

Algorithm 4 Baseline Detection and Mitigation

1: **procedure** BASELINE SCHEME
2: $T_h \leftarrow$ time horizon to observe
3: Linear map $\Phi^{(T_h)} : \mathbf{x} \rightarrow [H\mathbf{x} \mid HA_d\mathbf{x} \mid \cdots \mid HA_d^{T_h-1}\mathbf{x}]$
4: Form $Z^{(T_h)} = [\mathbf{z}[0] \mid \mathbf{z}[1] \mid \cdots \mid \mathbf{z}[T_h - 1]]$
5: Solve $\underset{\hat{\mathbf{x}} \in \Re^n}{\operatorname{argmin}} ||Z^{(T_h)} - \Phi^{(T_h)}\hat{\mathbf{x}}||_{\ell_1/\ell_2}$

where the matrix norm $||M||_{\ell_1/\ell_2}$ for any matrix M with p rows is defined as

$$||M||_{\ell_1/\ell_2} = \sum_{i=1}^{p} ||M_i||_{\ell_2}$$

where M_i is the i-th row of M.

Incorporating $\mathbf{u}(t)$ in the analysis along with the assumption of unattacked actuators, the linear map $\Phi^{(T_h)}$ in Algorithm 4 gets transformed to $\Phi^{(T_h)} : \mathbf{x} \rightarrow [H\mathbf{x} \mid H(A_d\mathbf{x} + B\mathbf{u}[0]) \mid H(A_d^2\mathbf{x} + A_dB_d\mathbf{u}[0] + B_d\mathbf{u}[1]) \mid \cdots \mid H(A_d^{T_h-1}\mathbf{x} + A_d^{T_h-2}B_d\mathbf{u}[0] + \cdots + B_d\mathbf{u}[T_h-1])]$. The optimization algorithm collects data until $t = T_h$. After that, the algorithm takes some time to converge to the correct state estimate $\mathbf{x}[0]$. Then the subsequent states till the present instant are computed using (6.33a) and the current updated state is utilized to compute the required control after a total of T_u time steps from $t = T_h$. Hence, the mitigation latency for the baseline scheme is defined as $T_b = T_h + T_u$.

In presence of actuator attacks, the $\mathbf{u}(t)$ is unknown and the diagnosis and correction algorithm aims to estimate both the sensor and actuator attack sets K_y and K_x as well as reconstruct the sequence of states $\hat{\mathbf{x}}[0], \hat{\mathbf{x}}[1], \cdots, \hat{\mathbf{x}}[T_h - 1]$. In this case, the optimization

problem (6.35) is reformulated to:

$$\begin{aligned} \text{minimize} \quad & |\hat{K}_y| + |\hat{K}_x| \\ \text{subject to} \quad & \text{supp}(\hat{\mathbf{a}}_y[t]) \subseteq \hat{K}_y,\ \text{supp}(\hat{\mathbf{a}}_x[t]) \subseteq \hat{K}_x \\ & \mathbf{z}[t] = H\hat{\mathbf{x}}[t] + \hat{\mathbf{a}}_y[t] \\ & \hat{\mathbf{x}}[t+1] = A_d\hat{\mathbf{x}}[t] + B_d(\mathbf{u}[t] + \hat{\mathbf{a}}_x[t]) \\ & \text{for} \quad t \in \{0, 1, \cdots, T_h - 1\}. \end{aligned} \tag{6.36}$$

As previously discussed, the ℓ_0-norm optimization problem is NP-hard and replaced by an ℓ_1-norm. This relaxation leads to a tractable optimization problem[188] that can be solved using convex optimization routines. It must be noted that in the previous case only the initial state $\mathbf{x}[0]$ needs to be reconstructed since the known actuator inputs along with the dynamical equations are used to generate all the subsequent states and compute the corrective action. For actuator attacks, the aim is to reconstruct the entire sequence of states. Compensatory action is difficult to apply in presence of actuator attacks and preventive measures such as isolation of attacked nodes, identification of communication breach or temporary shutdown are undertaken. A schematic flow of the baseline scheme is shown in Figure 6.10.

6.7.2 Check-assisted mitigation

The check-assisted scheme is proposed to reduce the latency of attack diagnosis and correction compared to the baseline scheme. The basic premise of the proposed scheme is to reduce the search space of the attack sets in the convex optimization algorithm by isolating the sensor and actuator node blocks under attack. This is achieved by executing fast binary search algorithms on the error signal computation in (6.24). At the instant of attack detection, redefined as time $t = 0$, the algorithm executing on the central control module, divides the check state transmissions from N_b local blocks into two groups - the first one containing blocks 1 to $\lfloor \frac{N_b}{2} \rfloor$ and the other one containing blocks $\lfloor \frac{N_b}{2} \rfloor + 1$ to N_b. At time $t = 1$, the

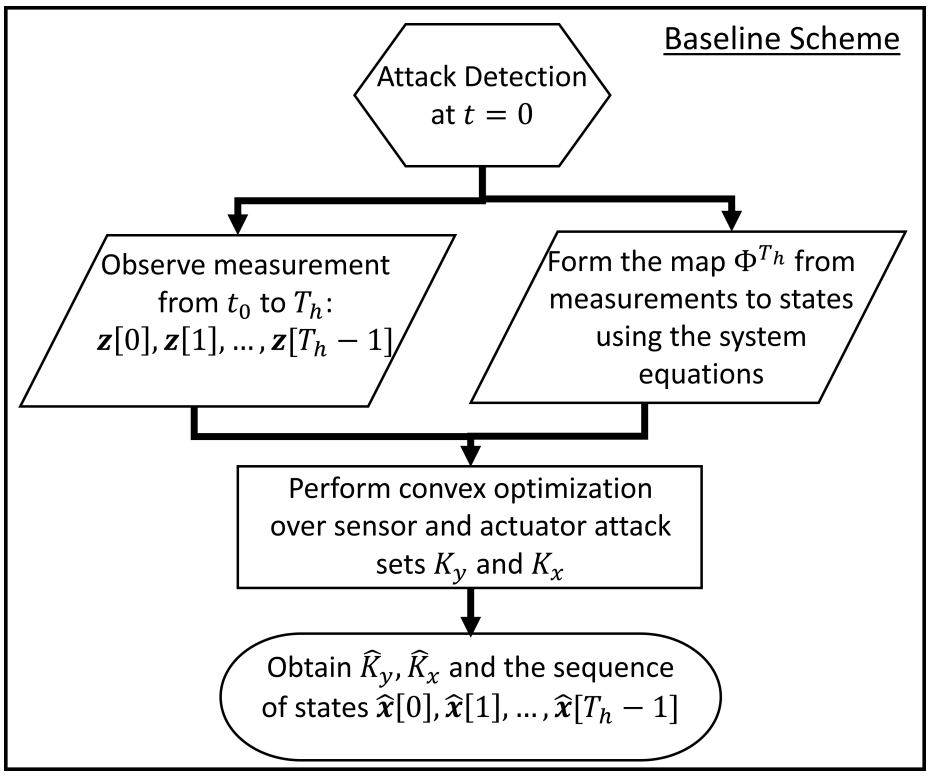


Figure 6.10: Flowchart of the baseline scheme presented in Algorithm 4

algorithm computes two separate error signals $e_1(t)$ and $e_2(t)$ from the two groups defined above as:

$$e_1 = \sum_{j=1}^{\left\lfloor \frac{N_b}{2} \right\rfloor} w_j.(\boldsymbol{\kappa}_j(*)\mathbf{s}_j).\hat{\mathbf{x}}_d - \sum_{j=1}^{\left\lfloor \frac{N_b}{2} \right\rfloor} e_{l(j)} \tag{6.37}$$

$$e_2 = \sum_{j=\left\lfloor \frac{N_b}{2} \right\rfloor + 1}^{N_b} w_j.(\boldsymbol{\kappa}_j(*)\mathbf{s}_j).\hat{\mathbf{x}}_d - \sum_{j=\left\lfloor \frac{N_b}{2} \right\rfloor + 1}^{N_b} e_{l(j)} \tag{6.38}$$

If sensors or actuators in one of the N_b blocks are under attack, then one of the error signals e_1 and e_2 is zero while the other is non-zero. The non-zero error signal localizes the attack within half of the total blocks. In the next time instant, the group containing the compromised block is again divided into 2 different sub-blocks and the algorithm proceeds until the attacked block is identified. This significantly reduces the potential attack set size

from the entire p sensor nodes to $|b_i|$ nodes, assuming that the i-th block is under attack (where b_i is the measurement set of the ith block as described in Section 6.6.2. However, non-zero values of both e_1 and e_2 at time $t = 1$ indicate that sensors and actuators from multiple blocks are under attack. In such a situation, at time $t = 2$, two concurrent binary search algorithms are executed on both the groups for further attack localization. The flowchart for the check-assisted mitigation scheme is shown in Figure 6.11 and the binary search algorithm for isolating the attacked blocks is provided in Algorithm 5.

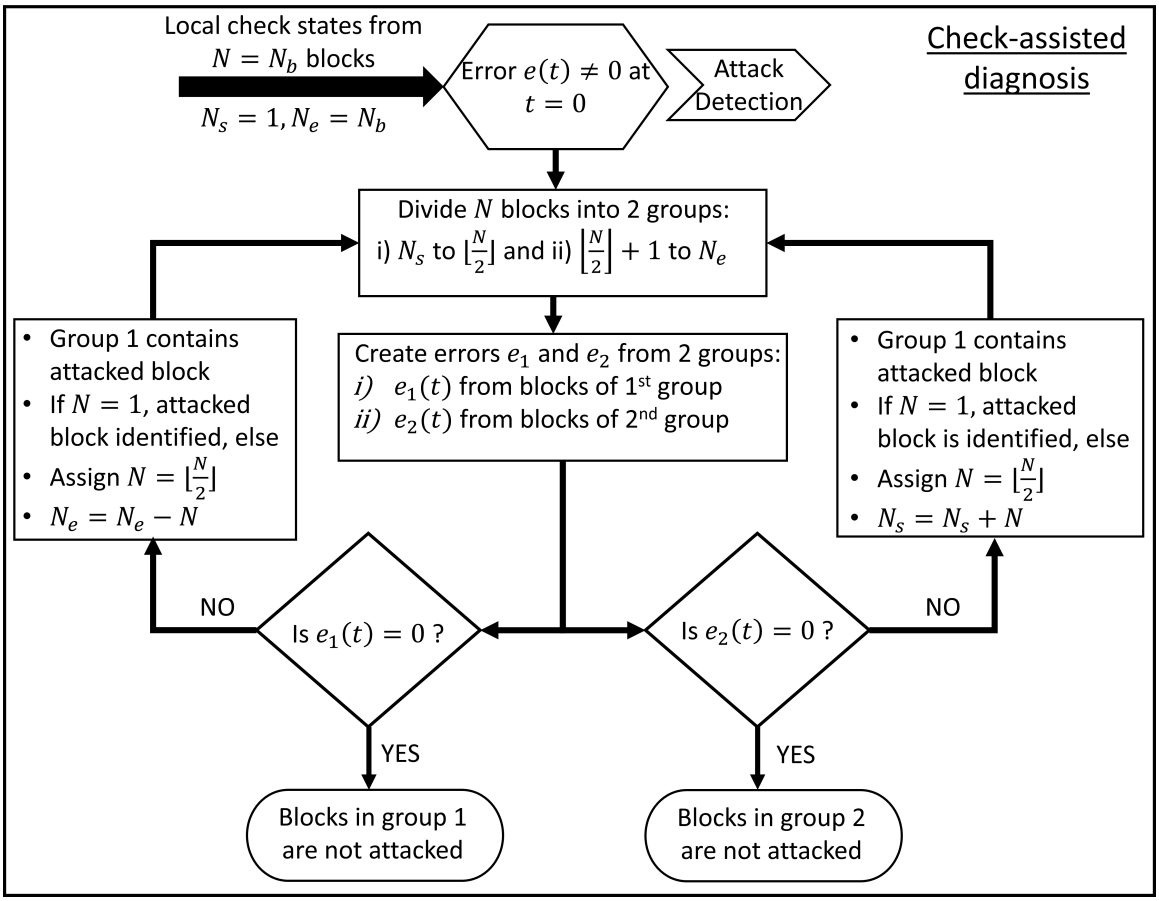


Figure 6.11: Flowchart of the check-assisted diagnosis scheme presented in Algorithm 5

From the concept of strong observability in linear systems[201, 202], it has been proved in [188] that the maximal number of attacks that the optimization algorithm is guaranteed to compensate is $\left\lceil \frac{p}{2} - 1 \right\rceil$. If the number of attacked measurements is higher than that, the optimization algorithm may not be able to correctly estimate the system states even

Algorithm 5 Check-assisted diagnosis

procedure BINARY SEARCH
  $N_b \leftarrow$ number of blocks
  $\mathbf{x}_d \leftarrow$ distant state estimate
  $e_{l(j)} \leftarrow$ local j-th block error
  $n_s = 1, n_e = N_b, n_d = \left\lfloor \frac{N_b}{2} \right\rfloor$
  $t = 1$
  while $n_d \geq 1$ **do**
    $n_1 = n_s + n_d - 1$
    if $(n_e - n_s + 1) \bmod 2 = 0$ **then**
      $n_2 = n_e - n_d + 1$
    else
      $n_2 = n_e - n_d$
    $e_1 = \sum_{j=n_s}^{n_1} w_j.(\boldsymbol{\kappa}_j(*)\mathbf{s}_j).\hat{\mathbf{x}}_d - \sum_{j=n_s}^{n_1} e_{l(j)}$
    $e_2 = \sum_{j=n_2}^{n_e} w_j.(\boldsymbol{\kappa}_j(*)\mathbf{s}_j).\hat{\mathbf{x}}_d - \sum_{j=n_2}^{n_e} e_{l(j)}$
    if $e_1 = 0, e_2 \neq 0$ **then**
      $n_s = n_s + n_d$
    else if $e_1 \neq 0, e_2 = 0$ **then**
      if $(n_e - n_s + 1) \bmod 2 = 0$ **then**
        $n_e = n_e - n_d$
      else
        $n_e = n_e - n_d - 1$
    else if $e_1 \neq 0, e_2 \neq 0$ **then**
      Do BINARY SEARCH concurrently on 2 groups
    $n_d = \left\lfloor \frac{n_e - n_s + 1}{2} \right\rfloor$
    $t = t + 1$
  Attacked block $\leftarrow n_s$

if the time horizon T_h is increased to include more measurements. The optimal choice of T_h depends on the number of attacked sensor and actuator nodes with the maximum value of $T_{h(max)} = n$. With the increase in number of attacked nodes, more measurements are required to successfully recover the system states. Algorithm 5 significantly reduces the possible attack set size in $T_d = \log_2 N_b$ time steps. This reduces the number of measurements T_h needed by the optimization algorithm to converge. Suppose, the reduced time horizon is T_{hr}. The binary search execution time $T_d << T_{hr}$ because $T_d = \log_2 N_b < N_b << n$ and $T_{hr} \sim \mathcal{O}(n)$. The reduction in search space for the op-

timization program reduces its convergence time from T_u to T_{ur}. Hence, the mitigation latency of the proposed checksum-assisted methodology is $T_p = T_{hr} + T_{ur}$. The reduction in mitigation latency of the checksum-assisted method from that of the baseline scheme is illustrated in Figure 6.12.

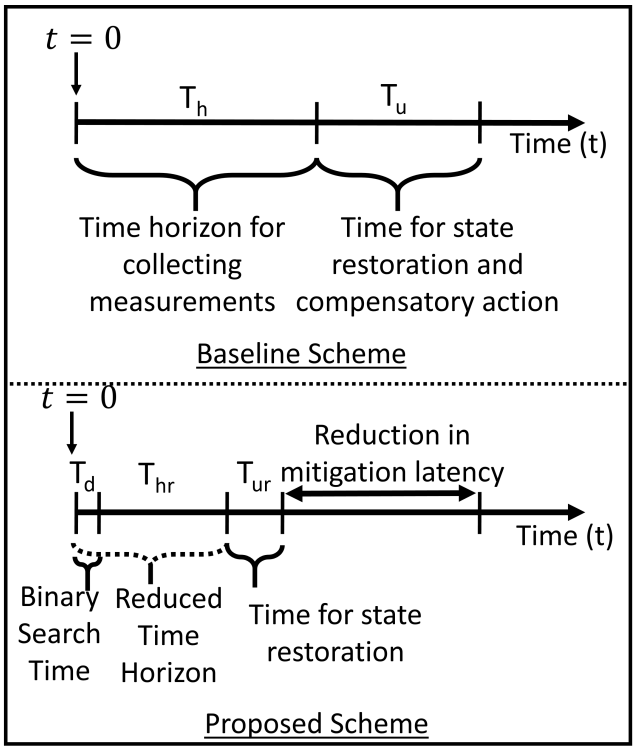


Figure 6.12: Temporal events after error detection for baseline and proposed schemes

6.7.3 Implementation and Overhead

The implementation of the proposed scheme requires local and secure estimation of states enabled by hardware links to the actual grid. This can be achieved by 2 ways:

1. *Secure Hardware* - A secure hardware implementation consists of FPGA modules that perform state estimation based on the received data along with appropriate communication interfaces for data transmission and reception. Prior works have demonstrated FPGA implementation of Kalman filters[203, 204] and extended Kalman filters[205, 206]. The additional state encoding functions are also implemented on the

same FPGA modules.

2. *Secure Software* - The local state estimation can also be performed by software in a highly secure environment close to the grid sensors such that external attacks are unable to penetrate such defenses. The central software module is still remotely located for coordinating the control of the entire grid.

Our detection algorithm computes a single checksum from n state-estimates with complexity $\mathcal{O}(n)$. The checksum assisted diagnosis method in Algorithm 5 assumes the knowledge of N_b and converges in $\log N_b$ time steps with $\mathcal{O}(\log_2 N_b)$. The number of blocks $N_b < n$ and hence the total complexity of our method is estimated as $\mathcal{O}(n)$. This is significantly lower than $\mathcal{O}(n^k)$ as reported in [186] where k is the attack set cardinality.

6.8 Simulation Results

We conduct experiments on the IEEE 5-bus, 9-bus, 14-bus, 30-bus, 118-bus and the 300-bus networks. For simulation of large distributed systems, we use the 1354-bus system from the Pan European Grid Advanced Simulation and State Estimation (PEGASE) project[207, 208] and the 1951-bus very high voltage system of the French Transmission Network[207]. The MATPOWER package is used to extract the network configuration and for solving the optimal DC power flow problems. For each test system, the state variables are the voltage angles at all the buses except one that is chosen as the reference bus with angle of 0°. Thus, for an $n-$bus network the number of state variables is $(n-1)$. The measurements include real power flows in all branches and real power injections at each bus (withdrawn load power is represented as negative power injection). The attacks are simulated by adding the attack vector to the measurement vector.

6.8.1 Attack Generation

From Section 6.5.2, it is clear that the replay attack is more potent due to its ability of directly disrupting the system dynamics through actuator injections. However, stealth attacks are more common because an adversary can indirectly affect normal system operation by compromising a small subset of the sensors. Hence, we analyze the difficulty in generation of stealth attacks from an adversarial perspective. We quantify the ease of attack by the metric *success probability*[172]. For generating the data, we vary the number of attacked nodes, say l from 1 to number of sensors (for example, this ranges between 1 to 711 for 300-bus system). For each choice of l, we randomly choose l sensors to compromise and execute Algorithm 6 to check if a valid attack exists. We perform this experiment 10000 times for all systems and compute the *success probability* p_l as $p_l = \frac{\text{\# successful trials}}{\text{\# total trials}}$. The success probability is plotted against the compromised sensor percentage in Figure 6.13 where it is seen that the success probability of attack generation increases with the proportion of attacked sensors.

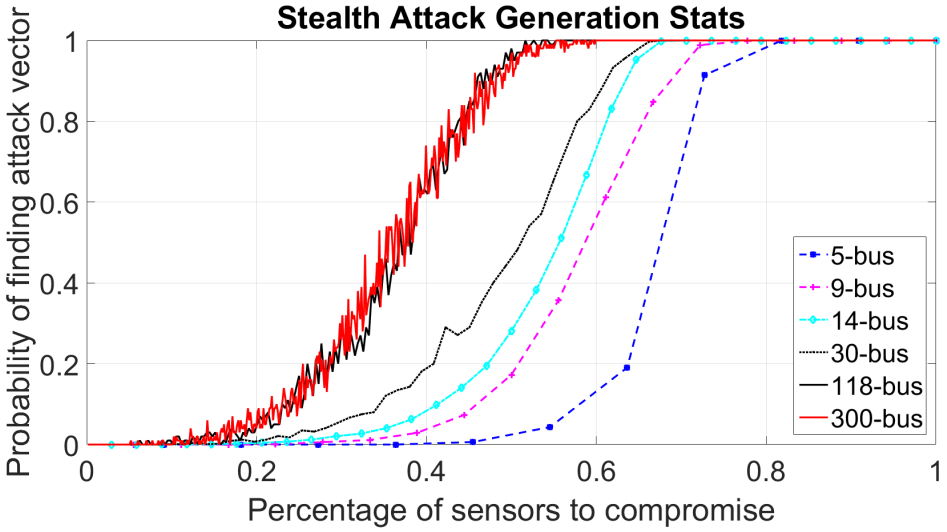


Figure 6.13: Plot of attack generation probability vs proportion of sensor nodes to attack across different IEEE benchmark test systems

From Figure 6.13, we emphasize on three points. Firstly, there exists a certain threshold percentage of total sensors that need to be compromised for successfully generating a

stealth attack. If an attacker manages to attack fewer sensors than this threshold, a stealth attack vector doesn't exist. Secondly, it can be noticed that this threshold reduces with the increase in number of buses evident from the shift of the curves towards left. This interesting observation is explained by the fact that the number of measurements increases with the number of buses and the absolute number of sensors that need to be compromised also increases despite the reduction in the threshold percentage. Finally, this analysis demonstrates that if a certain percentage of sensors are protected with additional security methods, stealth attacks cannot be generated by an attacker easily.

6.8.2 Centralized Attack Detection

For centralized error detection, we perform injection experiments for the two attack models on all the IEEE bus systems. Changes in load demands across the load buses are simulated. At the chosen instant of attack injection, the measurement vector $\mathbf{y}(t)$ is replaced with $\mathbf{y}(t) + \mathbf{a}_y(t)$, with $\mathbf{a}_y(t)$ being the sensor attack vector. In case of covert attack, the actuator attack vector $\mathbf{a}_x(t)$ is also added to the control signal.

Figure 6.14 illustrates stealth attack in 30-bus system. Three load changes are introduced in load buses 7, 21 and 30 at $t = 2.5s, 13s$ and $25.5s$ (indicated by black arrowheads). The top subfigure plots the actual voltage angle of bus 16 and its remote estimate from the measurements under attack. The unattacked state is also plotted in blue dashes as comparative reference. Gaussian noise is added to the measurements (measurement noise) and the local checksum (transmission noise) with 30 dB SNR. The attack injection occurs between $t = 5s$ (indicated by red arrowhead) and $t = 30s$. From Section 6.5.2, it is clear that the stealth attack adds a fixed offset to the measurements, that is compatible with the output matrix H. The remote estimate starts deviating from the actual state trajectory at the attack injection instant. This causes incorrect control signals (power injections) to be applied to the system that further deteriorates the state trajectory. It is seen in the figure that with time, the actual system state is significantly corrupted compared to unattacked case.

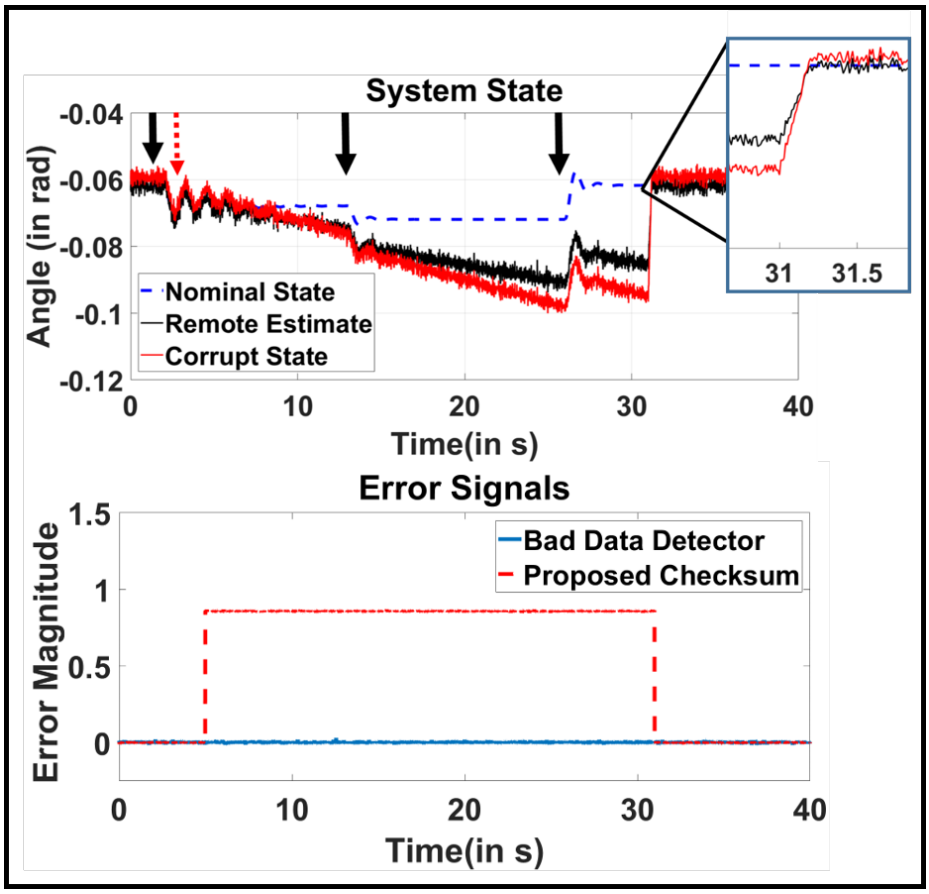


Figure 6.14: (Top) Nominal and corrupt state(voltage angle) along with remote estimate for bus 16 under stealth attack, (Bottom) Error values from the bad data detector and the proposed checksum technique

The remote estimate is different from the actual system state due to the continuous stealth attack injection. On withdrawal of the attack injection at the end of attack period, the controller swiftly brings back the system state to nominal trajectory as seen in the inset. The remote estimate also starts tracking the actual system state. A finite attack period is shown to illustrate the controller response after withdrawal of intrusion, though practically an attacker, after gaining access to communication channel is unlikely to discontinue the attack. The bottom subfigure plots the error signals implemented using a bad-data detector and the proposed checksum methodology. The bad data detector fails to detect any intrusion during the attack injection period whereas the checksum error shows a distinct signature indicating security attacks.

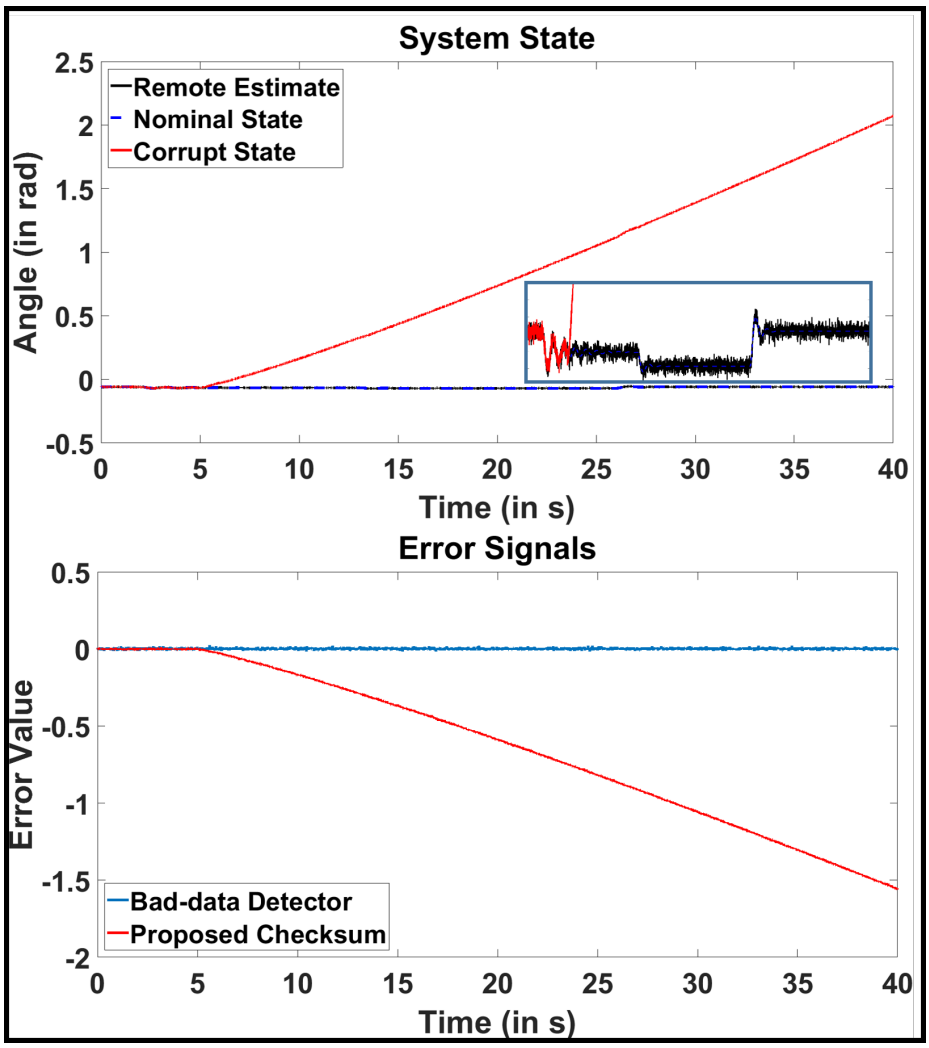


Figure 6.15: (Top) Nominal and corrupt state (voltage angle) along with remote estimate for bus 9 under replay attack, (Bottom) Error values from the bad data detector and the checksum methodology

Figure 6.15 demonstrates a replay attack scenario in the IEEE 30-bus system. At $t = 5s$, the sensor and actuator attack vectors are injected. The sensor attack is a recorded measurement vector of a previous nominal operation that is replaced instead of the actual measurement $\mathbf{y}(t)$. The attack vector can be any arbitrary signal with the objective of causing system damage. We choose ramp signals in the power injections of the generator buses 1, 2, 13, 22, 23 and 27 to demonstrate actuator attack. After attack injection, the actual system states start deviating from unattacked trajectory while the remote estimates are unable to track because of the corrupted sensor measurements. The top subfigure illustrates

the voltage angle drift of bus 9 to unstable values while the inset shows the remote estimate generating incorrect values from the modified measurements. The bottom subfigure demonstrates the error signal from the bad data detector and the checksum methodology. Since the local checksum tracks the compromised state and the error is generated as a difference between the local and remote checksum, the attack is instantly detected. In the 30-bus system, the measurement vector has a length of $p = 71$. The first three components of the pre-recorded measurement vector injected as the sensor attack is shown in Figure 6.16 to illustrate a replay attack vector.

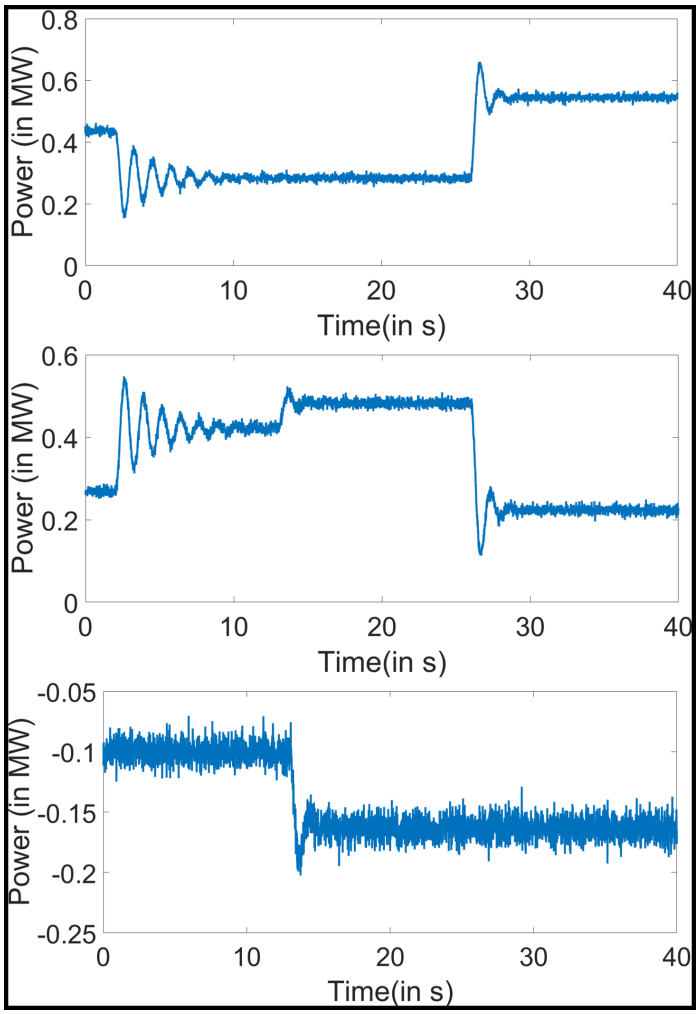


Figure 6.16: First three components of the replaced measurement vector in replay attack

6.8.3 Distributed Attack Detection

We demonstrate distributed attack detection on the 1354-bus system of the PEGASE project [207]. This high voltage European transmission network has 1354 buses, 260 generators, 1991 branches and 3344 measurements (1991 branch power flow measurements + 1353 bus power injections), with bus number 640 defined as the reference bus. 673 of the 1354 buses are load buses. We divide up the 3344 measurements into $N_b = 15$ blocks with 14 of the blocks containing 223 measurements and the other containing 222 measurements (since the power injection of the reference bus is known). The indexes and the number of measurements to be included are practically determined by the physical topology and geographical location of the grid infrastructure. In our experiments, we choose equal number of measurements in each of the blocks since the effectiveness of the distributed checksum scheme is not dependent on the block arrangements. Each of the 15 blocks receives a subset of the 3344 measurements and estimates a subset of the 1353 state variables. For example, block 2 estimates 184 states from 223 measurements. Each block computes its own checksum and transmits it back to the central control module. We choose linear encoding resulting in 15 different coding vectors for each of the 15 blocks. The distant encoding function $f_d(.)$ is determined from (6.31) with equal weight vector $\mathbf{w} = [1, 1, \cdots, 1]$ for function $f_e(.)$. For demonstration, we simulate load changes in 3 of the 673 buses introduced at $t = 1.5$s, 3.8s and 7.5s. At each sampling instant, the network variables are solved using the DC optimal power analysis of MATPOWER package. The 15 local checksum encodings are illustrated in Figure 6.17.

It is seen that most of the checksum codes have fixed magnitude while a few have transient profiles. Since we simulated load changes in only 3 of the load buses, most of the system states are constant and voltage angles of a few buses change as load demands are altered. Under unattacked situation, the distant checksum code matches the sum of the local checksum codes (since the weight vector $\mathbf{w}$ is chosen to be all 1s). However, the final error signal is not exactly zero due to the injection of transmission noise with 30 dB SNR. For

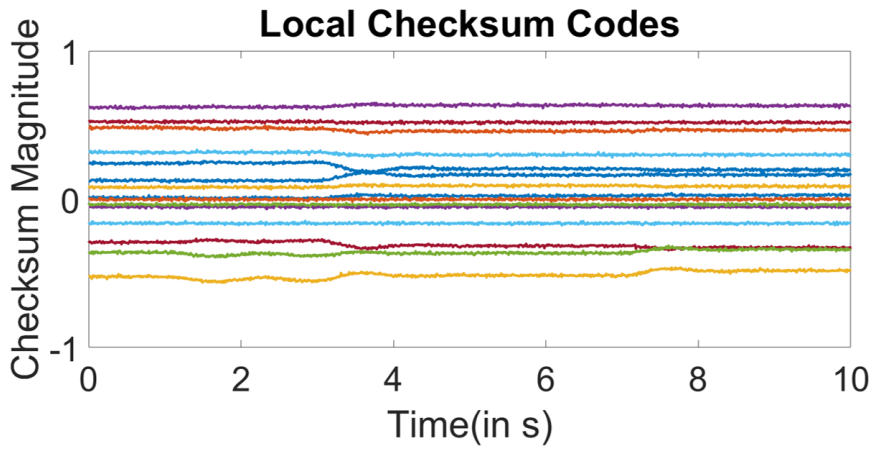


Figure 6.17: Local checksum codes from the $N_b = 15$ blocks for the 1354-bus system

attack demonstration, we introduce a stealth attack between $t = 1.5$s and $t = 7.8$s where sensors in block 4 are compromised. This results in the corruption of the distant estimate and subsequently the distant checksum code. After attack injection, the signal power of the error $e(t)$ rises higher than the noise floor and generates a clearly discernible signature as shown in Figure 6.18.

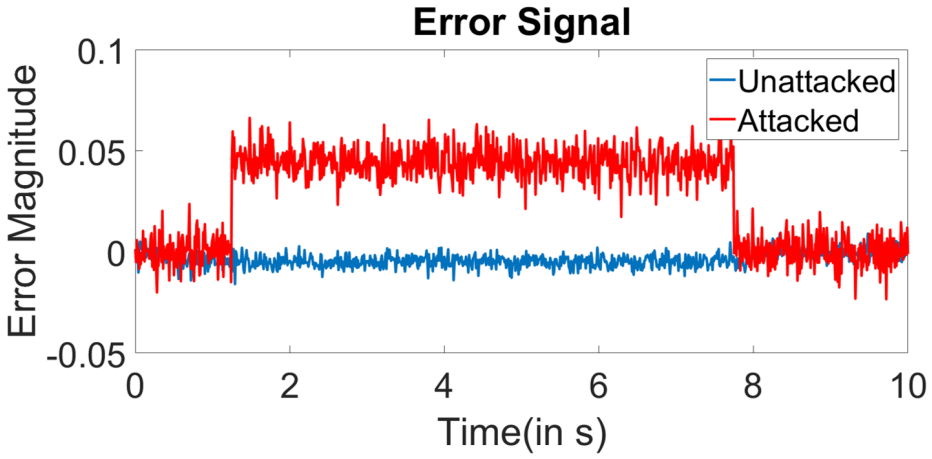


Figure 6.18: Error signal $e(t)$ with and without attacks for the 1354-bus system

6.8.4 Effects of Measurement Noise

To explore the effects of measurement noise on the detection accuracy, we vary both the measurement and transmission noise in our simulation of the IEEE benchmark systems.

As discussed in Section 6.8.4, the detection performance is not affected by measurement noise and depends on transmission noise only. In Figure 6.19, we demonstrate a situation where an attacker is able to conceal his intrusion because the attack signature stays below the noise floor of the error signal.

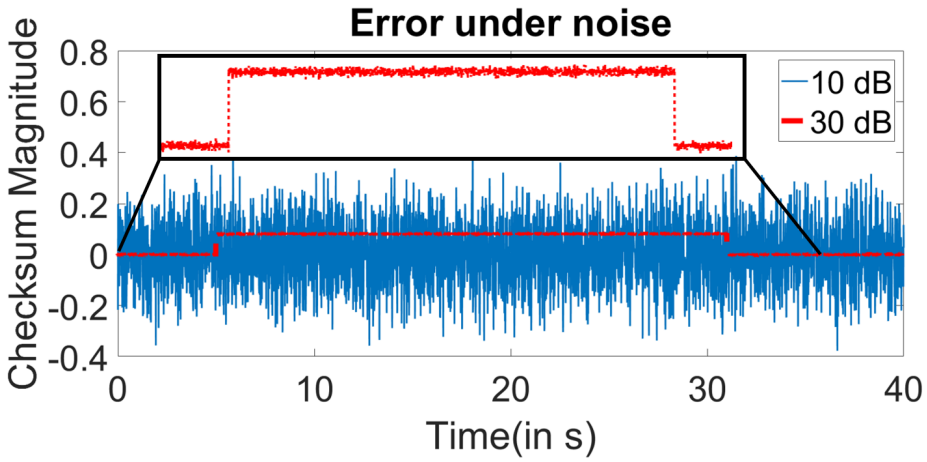


Figure 6.19: Error signal with variation in transmission noise statistic with 10 dB and 30 dB SNR clearly shows how an attack can go undetected with high noise floor

We simulate a stealth attack scenario in the IEEE 57-bus network. We inject an attack vector by compromising 69 of the 137 sensors. We inject Gaussian distributed measurement noise and transmission noise, both with mean 0 and variance 0.001, resulting in 30 dB SNR. The inset of Figure 6.19 illustrates the attack detection through prominent error signature. Next, retaining all choices, we increase the transmission noise variance to 0.05 such that the SNR is reduced to 10 dB. We notice that the higher noise floor of the error $e(t)$ completely subsumes the attack signature causing detection failure. So, we conduct 1500 simulations of the IEEE 57-bus system to compute attack coverage variation with error SNR. We execute 100 simulations for 15 different noise variances as shown in Figure 6.20. The threshold of attack detection varies with the noise floor of the error signal. We define an attack to be successfully detected if the absolute value of error lies above the chosen threshold during the attack injection period. *Attack coverage* a_c is defined as $a_c = \frac{\text{\# of detections}}{\text{\# of injections}}$. The variation of a_c with error SNR is shown in Figure 6.20.

From Figure 6.20 it is seen that the attack detection coverage drops to 0 for SNR $<$ 10

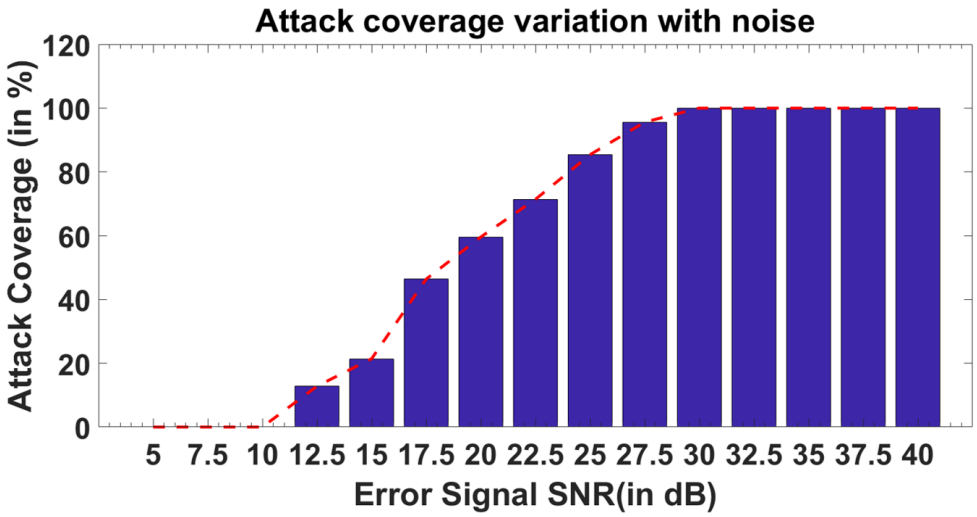


Figure 6.20: Variation of attack detection coverage with SNR of error signal

dB. From (6.32) we know that 10 dB indicates equal signal and noise power that prevents isolation of any attack signature above the noise floor in the error signal. A possible solution to increased noise statistic involves modifying the local and distant checksum codes to generate higher signal power, thus recovering the SNR of error signal. However, a structured algorithm to sense the noise statistic and tune these encodings has not been pursued in this work and will be studied in future research.

6.8.5 Attack Mitigation

For attack mitigation experiments, we solve the optimization problems using CVX[209] - an open-source MATLAB tool for disciplined convex programming. We demonstrate the attack mitigation procedure on the 1951-bus very high voltage French transmission network. This network has 1951 buses, 366 generators, 1010 load buses, 2596 branches and 4547 measurements. The sensor measurements are divided up into 20 blocks - 19 blocks with 227 measurements and 1 block having 234 measurements. Similar to the distributed detection scenario, the measurement assignment to the blocks depends on the geographical location of the sensors in reality and without this data, we allocate equal number of sensor measurements to each block. Both the measurement and the transmission noise variances

are set at 0.001 resulting in 38 dB and 32 dB SNR for the measurement and the error signals. First, we generate a stealth vector with 450 sensors ($\approx 10\%$) under attack. We inject the attack into the measurement vector at $t = 5$s. Load perturbations are introduced in 5 of the 1010 load buses at $t = 1$s, 11s, 16s, 20s and 31s for illustrative purpose. The voltage angle profile of bus 29 is illustrated in Figure 6.21.

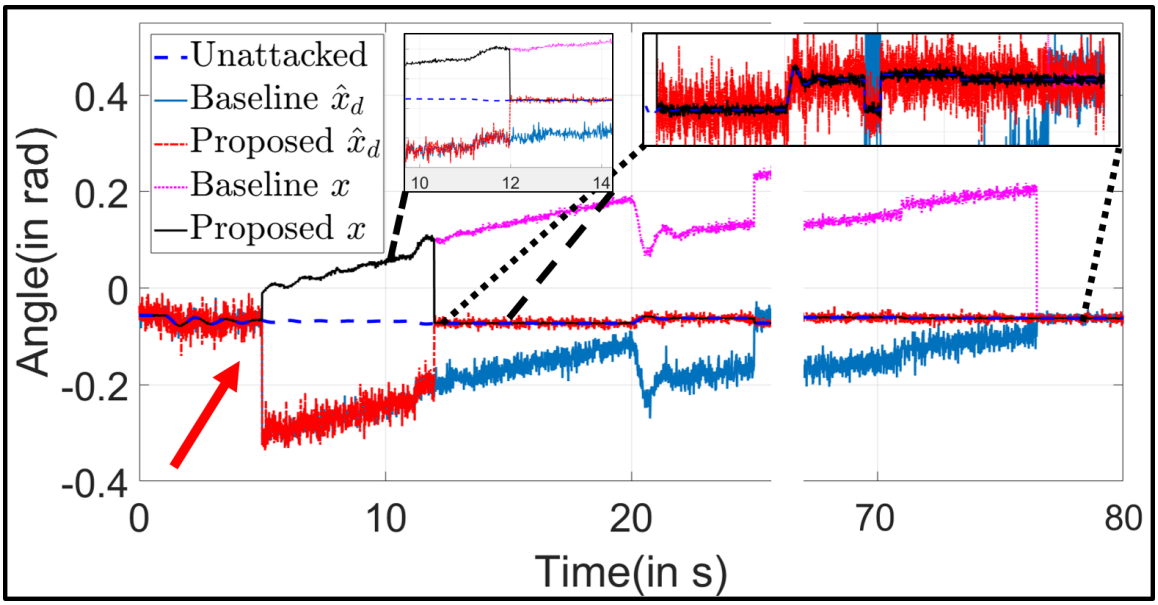


Figure 6.21: Voltage angle profile of bus 29 clearly shows the fast mitigation achieved in the proposed scheme in comparison with the baseline method

In our simulation, the sampling duration is fixed at $\delta t = 0.01$s. At $t = 5$s, the remote estimate $\hat{\mathbf{x}}_d(t)$ reduces sharply due to the injected attack, thus misguiding the generator injections and corrupting the system state $\mathbf{x}(t)$ that starts rising. Upon detection of error, Algorithm 5 is invoked and it isolates the plausible attack set to 3 blocks of 681 sensors ($\approx 15\%$ of total sensors) at $t = 5.05$s. The original time horizon is fixed at $T_h = 1950$ time steps. The reduced attack set cardinality allows us to decrease the required time horizon to $1950 \times 0.15 = 292$ time steps. Thus our proposed scheme collects transient data till $t = 5 + 2.92 = 7.92$s. Next, the optimization algorithm converges at $t = 11.98$s. Upon attack set diagnosis, the remote estimate is exactly able to start tracking the system state from $t = 11.98$s onwards and the power injections are able to restore the bus angle to

unattacked trajectory at $t = 12.56$s. In comparison, the baseline scheme collects data till $t = 5 + 19.5 = 24.5$s and the baseline algorithm operates on the entire search space of 4547 sensors for diagnosis. The baseline scheme converges at $t = 65.88$s and restores the system performance at $t = 66.48$s. The x-axis is shown discontinued for demonstration of all transient events.

For quantifying the benefits of the proposed methodology compared to the baseline scheme, we introduce a metric named *Optimization Convergence Rate* (*OCR*) that is defined as $OCR = \frac{T_b}{T_p}$, where T_b and T_p are defined in Section 6.7 as the mitigation latencies of the baseline and proposed scheme respectively. For example, in the above simulation, the baseline scheme takes $T_b = 6648 - 500 = 6148$ time steps for optimization convergence and the proposed scheme achieves the same in $T_p = 1256 - 500 = 756$ time steps, then this is quantified by $OCR = \frac{6148}{756} = 8.13$. This implies that the proposed methodology is 8.13X as fast as the baseline scheme. Thus the OCR provides a metric for envisioning the quantitative benefits of our work. The OCR indicates how fast a system is able to recover its nominal performance under ongoing attacks.

Figure 6.22 depicts the OCR data from our simulation results. We perform attack injections on 6 different systems - 30-bus, 57-bus, 114-bus, 300-bus, 1354-bus and 1951-bus networks. For each system, we divide the entire measurement into $N_b = 15$ different blocks with equal number of measurements (last block containing unequal number of measurements). For each system, we perform 250 simulations with 5 different sets of attacked sensors - 10%, 20%, 30%, 40% and 50% - 50 simulations with each set. In each simulation, we choose the attacked sensors and the load perturbations randomly. After the error signal $e(t)$ rises above the pre-defined detection threshold, both the baseline and our proposed scheme are deployed. As the proportion of attacked sensors is not known beforehand, the baseline scheme collects the transient data for the maximum time period of $T_h = n$ steps, where n is the number of states in the system. In our scheme, Algorithm 5 is invoked on error detection to isolate the group of attacked sensors. After Algorithm 5 restricts the

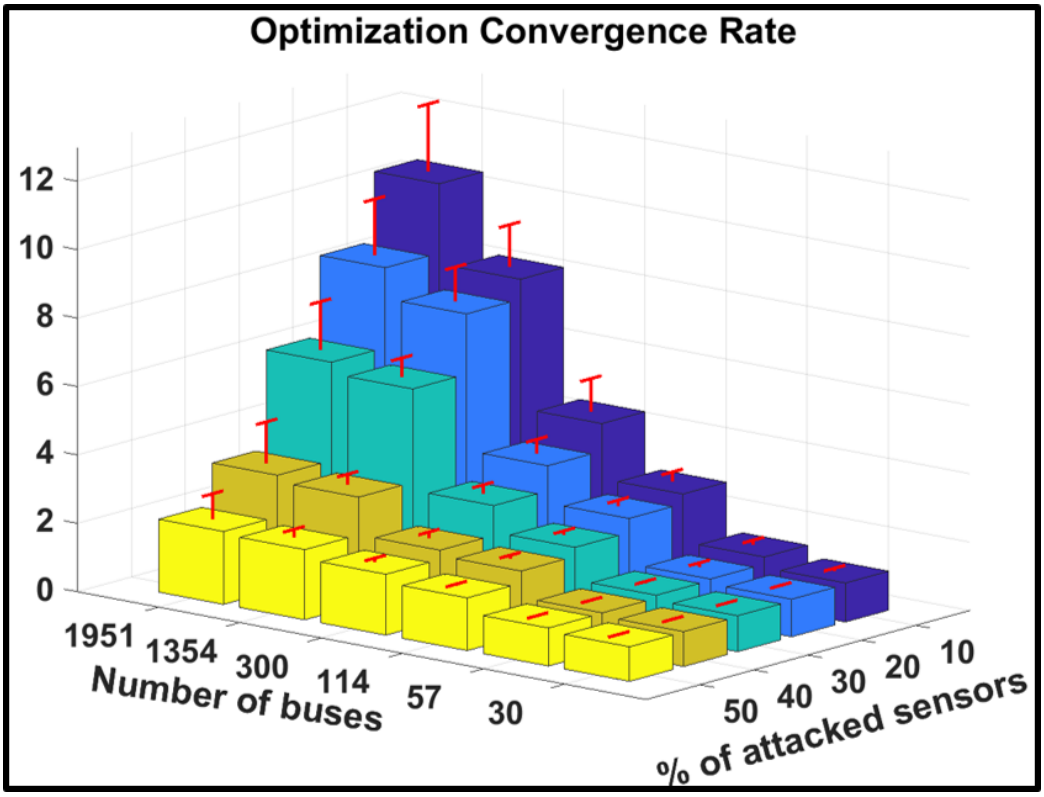


Figure 6.22: Optimization convergence rate for variations of attacked sensors in 6 different systems

possible attack set to selected sensors, the time horizon T_h is reduced to T_{hr} depending on the possible number of corrupt measurements as explained in the previous example. After the collection of measurement data over the selected horizon, the optimization is started in both the schemes for exact diagnosis of attack sets and obtain correct state estimates. In our scheme, the search space of the optimization algorithm is reduced to the diagnosed attack set from Algorithm 5. The final convergence times T_b and T_p of the baseline scheme and the proposed technique are recorded and used for OCR computation.

Figure 6.22 plots the mean OCR for all the experiments along with the standard deviations as the error bars. It is seen from the chart that OCR substantially grows with the number of buses. As the number of buses increases, the number of sensors grows rapidly that necessitates a high time horizon. On restricting the plausible attack set to a few measurements, our algorithm provides significant time savings in performing real-time

diagnosis. Another noticeable aspect is that the OCR decreases as the attack set cardinality increases. For example, in the 1951-bus system our scheme is 10.4X as fast as the baseline method with 10% attacked sensors while it drops to 2.13X for 50% of the attack sets. This is because under 50% of sensors under attack, the time horizon reduction from T_h to T_{hr} is absent and the only benefit we gain is from the reduced search space of the optimization algorithm. These data conclusively demonstrate the comprehensive benefits of our proposed work in detection, diagnosis and compensation of a power grid under security attacks.

6.9 Summary

In this work, we propose a hierarchical checksum based methodology for fast detection and diagnosis of attacks in cyber-physical systems. This scheme enables us to quickly recover the system performance in the presence of attacks and achieve minimum disruption. The extensive simulation results demonstrate the effectiveness of the proposed research in comparison to competing methods. In addition, we discuss the effects of different noise sources on our implementation, thus providing valuable insights into future efforts. Further, we investigate possible countermeasures from an attacker's perspective to jeopardize the efficacy of our proposed scheme.

CHAPTER 7
CONCLUSIONS AND FUTURE WORK

7.1 Conclusions

In this dissertation, a framework for real-time detection and correction of errors in linear and nonlinear systems is presented. The state-space encoding based checking methodology efficiently detects system errors due to component malfunctions in control systems as well as security attacks in cyber-physical systems.

Chapter 2 provided the theoretical basis of the state-space encoding and defined how a checksum is constructed that is capable of detecting abnormalities in system operation. The feedback of the constructed checksum error to the erroneous state was shown to accurately compensate the effects of faults on the system performance in real-time.

Chapter 3 demonstrated that state-space based encoding scheme can be used to cancel out transient noise signals. Two real-time learning approaches were proposed to detect the affected state for feedback of the checksum signal. Simulation results and hardware experiments on a Butterworth filter corroborated the compensating capabilities of the proposed scheme. In addition, a linear predictive coding based approach was presented for detection of transient soft errors in nonlinear Volterra filter operation.

Chapters 4 and 5 demonstrated how state-space encoding can be used to detect errors and recover system performance in real-time. For a general class of nonlinear control systems operating under arbitrary failure mechanisms, machine learning based checking methodologies were proposed and shown to detect errors across the different abstraction layers. Bit-flips in an ARM core implemented on FPGA demonstrated that actual soft errors in the digital processor are also detected through this scheme. Reinforcement learning based control was used as an augmentation to the nominal controller and real-time

self-learning was performed to recover system performance. The predictive check based error signal was intelligently exploited to bootstrap the learning process, thus reducing the latency of learning. Simulation results on two test cases and hardware experiments on an actual self-balancing robot showcased the benefits and effectiveness of the proposed schemes.

Chapter 6 presented a modified form of state-space encoding for attack resilience in distributed cyber-physical systems. The resilience of the proposed methodology to adversarial conuter-attacks was discussed along with the effects of noise on the attack coverage. Simulation results on the IEEE benchmark power systems and two European high voltage grids comprehensively illustrated the benefits and capabilities of the mitigation scheme.

7.2 Future Work

The future endeavors from this research can be summarized as:

1. The prediction functions (implemented by machine learning algorithms) for error checking in nonlinear control systems depend on the pre-deployment training with the expected system inputs during nominal operation. The training space for all possible inputs can be huge in a practical application and hence makes the pre-deployment training procedure almost infeasible. To circumvent this requirement, a *real-time learning* methodology needs to be adopted that learns these checking functions during the initial phase of nominal system operation after deployment. Such a scheme will relearn the checking functions after an error has been corrected such that future faults can be detected once again.

2. The real-time adaptation of the reinforcement learning parameters can jeopardize the safe operation of a system due to the episodic experiments that are needed. An important focus of future research need to concentrate on self-learning capabilities in safety-critical systems where it is imperative that the system is brought to a safe and

secure mode of operation before attempting to adapt in real-time. Efforts in this area will pave the way for the development of future self-healing autonomous systems.

3. The resilience of the cyber-physical system to external security attacks need to be extended to thwart malicious attacks on AC power flow analysis and system topology. Developing a comprehensive security framework for smart grids also need to consider malware based attacks on the control software present in the SCADA modules and propose low cost algorithm based checking and error recovery. Present work [210] have started focusing on real-time monitoring of control algorithms to check for any inconsistency induced by malware attacks or soft errors.

Appendices

APPENDIX A

HJB EQUATION

The optimal value function V^* for the optimal policy μ^* is defined as

$$V^*(\boldsymbol{x}(t)) = \max_{\boldsymbol{u}[t,\infty)} \left[\int_t^\infty e^{-\frac{s-t}{\tau}} \rho(\boldsymbol{x}(s), \boldsymbol{u}(s)) ds \right] \tag{A.1}$$

According to the optimality principle, the integral in (A.1) is divided into two parts $[t, t+\Delta t]$ and $[t+\Delta t, \infty)$ and then solve the subsequent optimization problem as,

$$V^*(\boldsymbol{x}(t)) = \max_{\boldsymbol{u}[t,\infty)} \left[\int_t^{t+\Delta t} e^{-\frac{s-t}{\tau}} \rho(\boldsymbol{x}(s), \boldsymbol{u}(s)) ds + e^{-\frac{\Delta t}{\tau}} V^*(\boldsymbol{x}(t+\Delta t)) \right] \tag{A.2}$$

For a small Δt, the first term is approximated as,

$\rho(\boldsymbol{x}(t), \boldsymbol{u}(t))\Delta t + o(\Delta t)$, and the second term is Taylor expanded as

$V^*(\boldsymbol{x}(t+\Delta t)) = V^*(\boldsymbol{x}(t)) + \frac{\partial V^*}{\partial \boldsymbol{x}(t)} f(\boldsymbol{x}(t), \boldsymbol{u}(t))\Delta t + o(\Delta t)$

Substituting these into (A.2) and collecting $V^*(\boldsymbol{x}(t))$ on the left-hand side, the optimality condition for $[t, t+\Delta t]$ is expressed as,

$$(1 - e^{-\frac{\Delta t}{\tau}}) V^*(\boldsymbol{x}(t)) = \max_{\boldsymbol{u}[t,t+\Delta t]} \left[\rho(\boldsymbol{x}(t), \boldsymbol{u}(t))\Delta t + e^{-\frac{\Delta t}{\tau}} \frac{\partial V^*}{\partial \boldsymbol{x}(t)} f(\boldsymbol{x}(t), \boldsymbol{u}(t))\Delta t + o(\Delta t) \right] \tag{A.3}$$

Dividing both sides by Δt and taking Δt to zero, the condition of the optimal value function is computed as,

$$\frac{1}{\tau}V^*(\boldsymbol{x}(t)) = \max_{\boldsymbol{u}(t)\in U}\left[\rho(\boldsymbol{x}(t),\boldsymbol{u}(t)) + \frac{\partial V^*}{\partial \boldsymbol{x}} f(\boldsymbol{x}(t),\boldsymbol{u}(t))\right] \tag{A.4}$$

Thus, the optimal policy is given by the action that maximizes the right-hand side of the HJB equation:

$$\boldsymbol{u}(t) = \arg\max_{\boldsymbol{u}\in U}\left[\rho(\boldsymbol{x},\boldsymbol{u}) + \frac{\partial V^*(\boldsymbol{x})}{\partial \boldsymbol{x}} f(\boldsymbol{x},\boldsymbol{u})\right] \tag{A.5}$$

APPENDIX B

VULNERABILITY ANALYSIS

In the early 1960s, Rudolph Kalman introduced the notions of controllability and observability as fundamental properties of linear systems. In this work, we extend the concept of observability/detectability to provide a metric of system vulnerability that is an inherent property of a system. We introduce a vulnerability index for both linear and nonlinear systems and provide the significance of such a metric later.

B.1 Linear Systems

A dynamic model for a linear, time-invariant system is represented as,

$$\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + B\mathbf{u}(t) \tag{B.1}$$

$$\mathbf{y}(t) = C\mathbf{x}(t) \tag{B.2}$$

where $\mathbf{x} \in \Re^n$, $\mathbf{u} \in \Re^m$ and $\mathbf{y} \in \Re^p$ are the system state, control input to the system and output/measurement variables, respectively. The notion of observability stems from the question whether an initial condition can be uniquely determined within a finite time from the system measurements. The linear observability matrix O is defined as,

$$O \doteq \begin{bmatrix} C \\ CA \\ CA^2 \\ . \\ . \\ CA^{n-1} \end{bmatrix} \tag{B.3}$$

If the np x n observability matrix has full rank implying rank(O) = n, then an initial state $\mathbf{x}_0$ in an unforced system ($\mathbf{u} = 0$) can be uniquely determined by the univalent mapping $\mathbf{x}_0 = (O^\intercal O)^{-1} O^\intercal \tilde{\mathbf{y}}$, where $\tilde{\mathbf{y}} = [\mathbf{y}, \dot{\mathbf{y}}, \cdots, \mathbf{y}^{(n-1)}]$. The full rank condition establishes complete system observability. A rank(O) $< n$ indicates the existence of non-observable modes in the system. Let us assume rank(O) = $k < n$. This implies that the dimension of the null-space of the observability matrix is $(n-k)$. One can find $(n-k)$ basis vectors to span $\mathcal{N}(O) \subset \Re^n$, the unobservable subspace of the system space. This implies that there exists a diagonalizing transformation T such that with change of variables $\bar{\mathbf{x}} = T\mathbf{x}$, the $(n-k)$ unobservable modes can be determined as $\bar{x}_1, \cdots, \bar{x}_{n-k}$. Hence, if an attacker manages to inject a malicious attack in the unobservable modes of the system, a state observer is unable to uniquely estimate the system state (since multiple solutions exist) resulting in incorrect control input and system trajectory deviation. In other words, unobservability in a system renders itself vulnerable to security attacks.

We define the vulnerability index $0 \leq \delta_v \leq 1$ as,

$$\delta_v = \frac{|\sigma_{min}(O^\intercal O)|}{|\sigma_{max}(O^\intercal O)|} \tag{B.4}$$

where σ_{min} and σ_{max} are minimum and maximum singular values of $O^\intercal O$ respectively.

$\delta_v = 1$ (all singular values are equal) indicates full observability and least vulnerability whereas $\delta_v = 0$ (presence of zero singular value implying non-zero null space) indicates no observability and most vulnerability. It is noteworthy that a value of $0 < \delta_v < 1$ still indicates full observability but strength of observability of all states are unequal. This implies that in presence of measurement noise, the confidence in estimating some states is higher than that of others. The system modes with low singular values of $O^\intercal O$ suffer from higher vulnerability and are potential targets for attackers.

B.2 Nonlinear Systems

Unlike linear systems, the nonlinear observability mapping is not constant and evolves over the state trajectory. Hence, the vulnerability index is a function of the phase space trajectory. An unforced nonlinear system ($\mathbf{u} = 0$) is represented as,

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) \tag{B.5}$$

$$y = h(\mathbf{x}) \tag{B.6}$$

Assuming that $h(.)$ is a sufficiently smooth scalar field on $\Re^n$ and taking consecutive derivatives up to order $(n-1)$, we get,

$$
\begin{aligned}
\dot{y} &= \frac{\partial h}{\partial \mathbf{x}}.\mathbf{f}(\mathbf{x}) \coloneqq \mathfrak{L}_f^1 h(\mathbf{x}) \\
y^{(2)} &= \frac{\partial (\mathfrak{L}_f^1 h)}{\partial \mathbf{x}}.\mathbf{f}(\mathbf{x}) \coloneqq \mathfrak{L}_f^2 h(\mathbf{x}) \\
&. \\
&. \\
&. \\
y^{(n-1)} &= \frac{\partial (\mathfrak{L}_f^{n-2} h)}{\partial \mathbf{x}}.\mathbf{f}(\mathbf{x}) \coloneqq \mathfrak{L}_f^{n-1} h(\mathbf{x})
\end{aligned}
\tag{B.7}
$$

Having formally defined the Lie derivatives, the differential embedding map $\phi : \Re \rightarrow \Re^n$ from measurement to the states is defined as,

$$
\phi(\mathbf{x}) = \begin{bmatrix} \mathfrak{L}_f^0 h(\mathbf{x}) \\ \mathfrak{L}_f^1 h(\mathbf{x}) \\ . \\ . \\ \mathfrak{L}_f^{n-1} h(\mathbf{x}) \end{bmatrix}
\tag{B.8}
$$

The nonlinear observability matrix is defined as the Jacobian of the map ϕ,

$$
O(\mathbf{x}) \equiv \frac{\partial \phi}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial \mathfrak{L}_f^0 h(\mathbf{x})}{\partial x_1} & . & . & \frac{\partial \mathfrak{L}_f^0 h(\mathbf{x})}{\partial x_n} \\ . & . & . & . \\ . & . & . & . \\ \frac{\partial \mathfrak{L}_f^{n-1} h(\mathbf{x})}{\partial x_1} & . & . & \frac{\partial \mathfrak{L}_f^{n-1} h(\mathbf{x})}{\partial x_n} \end{bmatrix}
\tag{B.9}
$$

Similar to linear systems, we again define a phase-trajectory dependent vulnerability

index $0 \leq \delta_v(\mathbf{x}) \leq 1$ as,

$$\delta_v(\mathbf{x}) = \frac{|\sigma_{min}(O^\intercal(\mathbf{x})O(\mathbf{x}))|}{|\sigma_{max}(O^\intercal(\mathbf{x}O(\mathbf{x}))|} \tag{B.10}$$

Unlike linear systems, the observability of a nonlinear system is not constant in the entire state-space over which it operates. The system is more vulnerable in a certain subspace $V \subset \Re^n$ where $V = \mathcal{N}(O^\intercal O)$ that is dependent on the phase trajectory. Hence, an attacker can track the system state and monitor the vulnerability index to inject an attack only when the phase trajectory passes through an unobservable subspace that renders itself more difficult to detect from the measurements.

B.3 Significance

The significance of a vulnerability index in systems is two-fold: i) firstly, it provides an evaluation of vulnerability in a system and creates an attack awareness for the system designers, and ii) secondly, it indicates the amount of resilience needed to thwart malicious attacks. This analysis is important for assessing the security aspect of any system and design appropriate counter-measures such as implementing additional state observations and improving the noise tolerance.

APPENDIX C

SECURITY ATTACKS

C.1 Stealth Attack

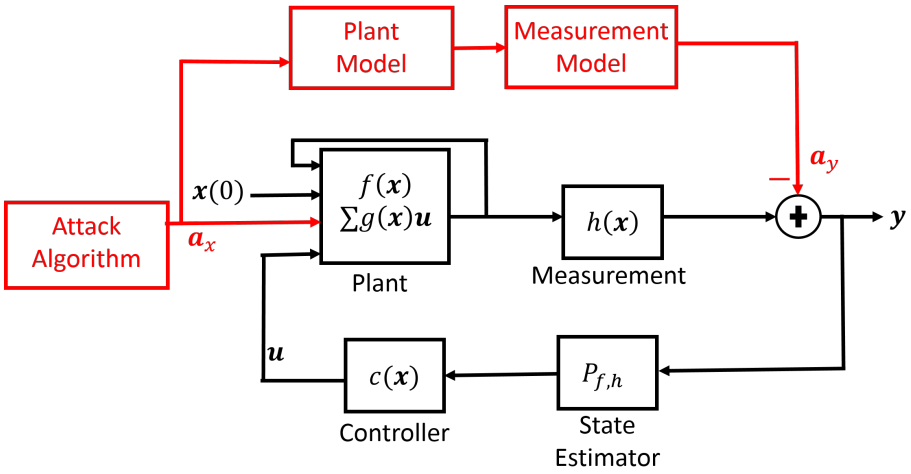
Algorithm 6 provides the mechanism to generate stealth attacks with a particular choice of attack set K_y.

Algorithm 6 Stealth Attack Generation

procedure STEALTH ATTACK
  $H \leftarrow$ measurement matrix
  $p \leftarrow$ number of sensors
  $l \leftarrow$ number of attacked sensor nodes
  Start with $l = 1$
  $K_x = \{k_1, k_2, \cdots, k_l\} \leftarrow$ attack set
  $\mathbf{a} = (0, \cdots, 0, a_{k_1}, 0, \cdots, 0, a_{k_2}, 0, \cdots, a_{k_l}, \cdots, 0)^\intercal$
  $P \leftarrow H(H^\intercal H)^{-1}H^\intercal$
  $B \leftarrow P - I_p$ with I_p: $p \times p$ identity matrix
  Let $B = (\mathbf{b}_1, \mathbf{b}_2, \cdots, \mathbf{b}_p)$ with $\mathbf{b}_i$ as ith column
  $\mathbf{a}' = (a_{k_1}, a_{k_2}, \cdots, a_{k_l})^\intercal$
  $B' \leftarrow (\mathbf{b}_{k_1}, \mathbf{b}_{k_2}, \cdots, \mathbf{b}_{k_l})$
  if rank$(B') = l$ **then**
    Non-zero attack vector does not exist
    Choose different K_x or increase l and execute
  else if rank$(B') < l$ **then**
    $B^\dagger \leftarrow$ pseudo-inverse of B'
    $\mathbf{d} \leftarrow$ Non-zero l-length vector
    $\mathbf{a}' \leftarrow (I_l - B^\dagger B')\mathbf{d}$
  Generate complete attack $\mathbf{a}$ from $\mathbf{a}'$

C.2 Covert Attack

Covert attack is the most potent form of attack where the attacker modifies the system measurements in such a way that the effect of its attack on the system dynamics is exactly canceled out as shown in Figure C.1. The model of a covert attack is $\mathbf{a} = [\mathbf{a}_x, -H(\tilde{\mathbf{x}})]^\intercal$,



b) Covert Attack

Figure C.1: Attack model of covert attack

where $\tilde{x}$ is the change in system states due to the injected attack. Covert attacks require the attacker to know the measurement topology, system dynamics and to have access to both sensor and actuator nodes, rendering this attack strategy to be the most potent among the three schemes discussed. This thesis has not explored schemes to tackle and detect covert attacks.

REFERENCES

[1] K. Krewell. “The slowing of Moore’s law and its impact”. In: *https://goo.gl/XVNbF1* (July 30, 2015).

[2] L. Eeckhout. “Is Moore’s Law Slowing Down? What’s Next?” In: *IEEE Micro* 37.4 (2017), pp. 4–5.

[3] Economist Quarterly. “After Moore’s Law”. In: *https://goo.gl/RpZZaq* (March 12, 2016).

[4] A. Oreskovic. “Intel admits the engine behind the tech industry’s amazing 50-year run is slowing down”. In: *https://goo.gl/TTwm5E* (March 23, 2016).

[5] T. Simonite. “Intel puts the brakes on Moore’s law”. In: *https://goo.gl/rpjBbP* (June 23, 2014).

[6] R. McMillan. “Is the end of Moore’s law slowing the world’s supercomputing race?” In: *https://goo.gl/bgUsWj* (June 23, 2014).

[7] P. Bright. “Moore’s law really is dead this time”. In: *https://goo.gl/YJEJbS* (February 2, 2016).

[8] J. Stokes. “Transistors go 3D as Intel re-invents the microchip”. In: *https://goo.gl/JL8U18* (May 4, 2011).

[9] P. Bright. “Moore’s law could stay on track with extreme UV progress”. In: *https://goo.gl/tKCpmv* (Aug 7, 2013).

[10] J. Deng and H. S. P. Wong. “A Compact SPICE Model for Carbon-Nanotube Field-Effect Transistors Including Nonidealities and Its Application - Part I: Model of the Intrinsic Channel Region”. In: *IEEE Transactions on Electron Devices* 54.12 (2007), pp. 3186–3194.

[11] S. Mitra. “Carbon nanotube imperfection-immune digital VLSI”. In: *International Symposium on Low Power Electronics and Design (ISLPED)*. 2013, pp. 144–144.

[12] B. Srinivasu and K. Sridharan. “Carbon nanotube FET-based low-delay and low-power multi-digit adder designs”. In: *IET Circuits, Devices Systems* 11.4 (2017), pp. 352–364.

[13] P. R. Y. Gangavarapu et al. “Graphene Electrodes as Barrier-Free Contacts for Carbon Nanotube Field-Effect Transistors”. In: *IEEE Transactions on Electron Devices* 64.10 (2017), pp. 4335–4339.

[14] B. C. Kang and T. J. Ha. “Single-Walled Carbon Nanotube Short-Channel Transistors Operating at Ultra-Low Voltages”. In: *IEEE Journal of the Electron Devices Society* 5.6 (2017), pp. 525–529.

[15] S. Banerjee et al. “Digitally-compatible ring oscillator frequency driven tuning of CN-TFT amplifiers: Performance compensation under statistical and morphological variations”. In: *2015 IEEE 20th International Mixed-Signals Testing Workshop (IMSTW)*. 2015, pp. 1–6.

[16] M. C. Lemme et al. “A Graphene Field-Effect Device”. In: *IEEE Electron Device Letters* 28.4 (2007), pp. 282–284.

[17] J. Kedzierski et al. “Epitaxial Graphene Transistors on SiC Substrates”. In: *IEEE Transactions on Electron Devices* 55.8 (2008), pp. 2078–2085.

[18] J. S. Moon et al. “Epitaxial-Graphene RF Field-Effect Transistors on Si-Face 6H-SiC Substrates”. In: *IEEE Electron Device Letters* 30.6 (2009), pp. 650–652.

[19] G. Fiori and G. Iannaccone. “Simulation of Graphene Nanoribbon Field-Effect Transistors”. In: *IEEE Electron Device Letters* 28.8 (2007), pp. 760–762.

[20] G. Liang et al. “Performance Projections for Ballistic Graphene Nanoribbon Field-Effect Transistors”. In: *IEEE Transactions on Electron Devices* 54.4 (2007), pp. 677–682.

[21] Y. Shiota et al. “Three-Terminal Device for Realizing a Voltage-Driven Spin Transistor”. In: *IEEE Transactions on Magnetics* 51.12 (2015), pp. 1–4.

[22] H. C. Koo, I. Jung, and C. Kim. “Spin-Based Complementary Logic Device Using Datta-Das Transistors”. In: *IEEE Transactions on Electron Devices* 62.9 (2015), pp. 3056–3060.

[23] J. S. Friedman et al. “Emitter-Coupled Spin-Transistor Logic: Cascaded Spintronic Computing Beyond 10 GHz”. In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 5.1 (2015), pp. 17–27.

[24] K. Konishi et al. “Gain and Fan-Out in a Current-Field Driven Spin Transistor With an Assisting AC Magnetic Field”. In: *IEEE Transactions on Magnetics* 48.3 (2012), pp. 1134–1138.

[25] S. Sugahara and J. Nitta. “Spin-Transistor Electronics: An Overview and Outlook”. In: *Proceedings of the IEEE* 98.12 (2010), pp. 2124–2154.

[26] D. Silver et al. “Mastering the game of Go with deep neural networks and tree search”. In: *Nature* 529.7587 (2016), pp. 484–489.

[27] W. Knight. “Five Lessons from AlphaGo’s historic victory”. In: *https://goo.gl/9GYQpQ* (Mar 18, 2016).

[28] G. Marcus. “Has Deepmind really passed Go?” In: *https://goo.gl/QT2y5W* (Jan 28, 2016).

[29] D. Milojicic. “What will our world look like in 2022?” In: *https://goo.gl/PSH7BK* (Sep, 2014).

[30] A. Sebastian et al. “Temporal correlation detection using computational phase-change memory”. In: *Nature Communications* 8.1115 (2017).

[31] G. Tosi et al. “Silicon quantum processor with robust long-distance qubit couplings”. In: *Nature Communications* 8.450 (2017).

[32] K. Bourzac. “IBM’s new chips compute more like we do”. In: *https://goo.gl/UxbU2R* (Ag 18, 2011).

[33] W. S. Levine, T. L. Johnson, and M. Athans. “Optimal limited state variable feedback controllers for linear systems.” In: *Automatic Control, IEEE Transactions on* 16 (1971), pp. 785–793.

[34] C. J. Wenk and C. Knapp. “Parameter optimization in linear systems with arbitrarily constrained controller structure”. In: *Automatic Control, IEEE Transactions on* 25.3 (1980), pp. 496–500.

[35] T. Soderstrom. “On some algorithms for design of optimal constrained regulators”. In: *Automatic Control, IEEE Transactions on* 23.6 (1978), pp. 1100–1101.

[36] V. Gupta, B. Hassibi, and R. M. Murray. “On the synthesis of control laws for a network of autonomous agents”. In: *American Control Conference, 2004. Proceedings of the 2004*. Vol. 6. IEEE. 2004, pp. 4927–4932.

[37] V. Gupta, B. Hassibi, and R. M. Murray. “A sub-optimal algorithm to synthesize control laws for a network of dynamic agents”. In: *International Journal of Control* 78.16 (2005), pp. 1302–1313.

[38] G. A. de Castro. “Convex methods for the design of structured controllers”. PhD thesis. University of California, Los Angeles, 2001.

[39] C. Langbort, R. S. Chandra, and R. D’Andrea. “Distributed control design for systems interconnected over an arbitrary graph”. In: *Automatic Control, IEEE Transactions on* 49.9 (2004), pp. 1502–1519.

[40] C. Langbort, V. Gupta, and R. M. Murray. “Distributed control over failing channels”. In: *Networked embedded sensing and control*. Springer, 2006, pp. 325–342.

[41] H. Kwakernaak and R. Sivan. *Linear optimal control systems*. Vol. 1. Wiley-Interscience New York, 1972.

[42] R. Hoseinnezhad and A. Bab-Hadiashar. “Fusion of Redundant Information in Brake-By-Wire Systems Using a Fuzzy Voter.” In: *J. Adv. Inf. Fusion* 1.1 (2006), pp. 52–62.

[43] R. Isermann, R. Schwarz, and S. Stolzl. “Fault-tolerant drive-by-wire systems”. In: *Control Systems, IEEE* 22.5 (2002), pp. 64–81.

[44] B. A. Guvenc and L Guvenc. “Robust steer-by-wire control based on the model regulator”. In: *Control Applications, 2002. Proceedings of the 2002 International Conference on*. Vol. 1. IEEE. 2002, pp. 435–440.

[45] W. Xiang et al. “Automobile brake-by-wire control system design and analysis”. In: *Vehicular Technology, IEEE Transactions on* 57.1 (2008), pp. 138–145.

[46] D. Skarin and J. Karlsson. “Software implemented detection and recovery of soft errors in a brake-by-wire system”. In: *Dependable Computing Conference, 2008. EDCC 2008. Seventh European*. IEEE. 2008, pp. 145–154.

[47] W. Hwang et al. “Model-based sensor fault detection algorithm design for electromechanical brake”. In: *Intelligent Transportation Systems (ITSC), 2011 14th International IEEE Conference on*. IEEE. 2011, pp. 962–967.

[48] J.l Slay and M. Miller. “Lessons Learned from the Maroochy Water Breach”. In: *Critical Infrastructure Protection*. Ed. by Eric Goetz and Sujeet Shenoi. Boston, MA: Springer US, 2008, pp. 73–82. ISBN: 978-0-387-75462-8.

[49] M. Abrams and J. Weiss. “Malicious control system cyber security attack case study-Maroochy Water Services, Australia”. In: *https://goo.gl/51npGm* (July 2008).

[50] R. J. Turk. “Cyber Incidents Involving Control Systems”. In: *https://goo.gl/cG5KxC* (October 2005).

[51] P. F. Roberts. “Zotob, PnP worms slam 13 DaimlerChrysler plants”. In: *https://goo.gl/P23Jtw* (August 2005).

[52] R. M. Lee, M. J. Assante, and T. Conway. “Media report of the Baku-Tbilisi-Ceyhan (BTC) pipeline cyber attack”. In: *https://goo.gl/gptyL4* (December 2014).

[53] D. Kravets. “Feds: Hacker disabled offshore oil platforms leak-detection system”. In: *https://goo.gl/uWfSGv* (Mar 2009).

[54] J. P. Farwell and R. Rohozinski. “Stuxnet and the Future of Cyber War”. In: *Survival* 53.1 (2011), pp. 23–40.

[55] D. Kushner. “The real story of Stuxnet”. In: *https://goo.gl/pbcf4e* (February 2013).

[56] N. Falliere, L. O. Murchu, and E. Chien. “W32.Stuxnet dossier”. In: *https://goo.gl/PTZdM9* (February 2011).

[57] R. M. Lee, M. J. Assante, and T. Conway. “German steel mill cyber attack”. In: *https://goo.gl/KsvktC* (December 2014).

[58] R. M. Lee, M. J. Assante, and T. Conway. “Analysis of the cyber attack on the Ukrainian power grid”. In: *https://goo.gl/QdaS3P* (March 2016).

[59] J. Von Neumann. “Probabilistic logics and the synthesis of reliable organisms from unreliable components”. In: *Automata studies* 34 (1956), pp. 43–98.

[60] E. F. Moore and C. E. Shannon. “Reliable circuits using less reliable relays”. In: *Journal of the Franklin Institute* 262.3 (1956), pp. 191–208.

[61] K. H. Huang and J. A. Abraham. “Algorithm-based fault tolerance for matrix operations”. In: *Transactions on Computers, IEEE* C-33 (1984), pp. 518–528.

[62] J. Y. Jou and J. A. Abraham. “Fault-tolerant matrix arithmetic and signal processing on highlyconcurrent computing structures”. In: *Proceedings of the IEEE* 74 (1986), pp. 732–741.

[63] R. Hegde and N.R. Shanbhag. “Soft digital signal processing”. In: *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* 9.6 (2001), pp. 813–823.

[64] N. Shanbhag. “Reliable and energy-efficient digital signal processing”. In: *Design Automation Conference, 2002. Proceedings. 39th*. 2002, pp. 830–835.

[65] S. Byonghyo and N.R. Shanbhag. “Energy-efficient soft error-tolerant digital signal processing”. In: *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* 14.4 (2006), pp. 336–348.

[66] S. Byonghyo and N.R. Shanbhag. “Reduced precision redundancy for low-power digital filtering”. In: *Signals, Systems and Computers, 2001. Conference Record of the Thirty-Fifth Asilomar Conference on*. Vol. 1. 2001, 148–152 vol.1.

[67] V.S.S. Nair and J.A. Abraham. “Real number codes for fault tolerant matrix operations on processor arrays”. In: *IEEE Transactions on Computers* 39.4 (April 1990), pp. 426–435.

[68] A. Chatterjee and M.A. d’Abreau. “The design of fault tolerant linear digital state variable systems: Theory and techniques”. In: *IEEE Transactions on Computers* 42.7 (July 1993), pp. 794–808.

[69] M. Ashouei and A. Chatterjee. “Checksum-Based Probabilistic Transient-Error Compensation for Linear Digital Systems”. In: *IEEE Transactions on VLSI* 17.10 (October 2009), pp. 1447–1459.

[70] M. Nisar and A. Chatterjee. “Guided Probabilistic Checksums for Error Control in Low Power Digital Filters”. In: *Transactions on Computer-Aided Design of Integrated Circuits and Systems, IEEE* (2011), pp. 1313–1326.

[71] A. Chatterjee and R.K. Roy. “Concurrent error detection in nonlinear digital circuits using time-freeze linearization”. In: *Computers, IEEE Transactions on* 46.11 (1997), pp. 1208–1218.

[72] M. I. Momtaz, S. Banerjee, and A. Chatterjee. “Probabilistic error detection and correction in switched capacitor circuits using checksum codes”. In: *2017 IEEE 23rd International Symposium on On-Line Testing and Robust System Design (IOLTS)*. 2017, pp. 271–276.

[73] A. Chatterjee. “Concurrent Error Detection and Fault Tolerance in Linear Analog Circuits Using Continuous Checksums”. In: *IEEE Transactions on VLSI* 1.2 (June 1993), pp. 138–150.

[74] H-G. D. Stratigopoulos and Y. Makris. “Concurrent Error Detection of erroneous responses of Linear Analog Circuits”. In: *Transactions on Computer-Aided Design of Integrated Circuits and Systems, IEEE* 25.5 (2006), pp. 878–891.

[75] S. Katayama, K. Yubai, and J. Hirai. “Iterative Design of the Reduced-Order Weight and Controller for the H_∞ Loop-Shaping Method Under Open-Loop Magnitude Constraints for SISO Systems”. In: *IEEE Transactions on Industrial Electronics* 56.10 (2009), pp. 3854–3863.

[76] S. Patra, S. Sen, and G. Ray. “Local stabilisation of uncertain linear time-invariant plant with bounded control inputs: parametric H_∞; loop-shaping approach”. In: *IET Control Theory Applications* 6.11 (2012), pp. 1567–1576.

[77] D. McFarlane and K. Glover. “A loop-shaping design procedure using H_∞ synthesis”. In: *IEEE Transactions on Automatic Control* 37.6 (1992), pp. 759–769.

[78] J. F. Whidborne, I. Postlethwaite, and D. W. Gu. “Robust controller design using H_∞ loop-shaping and the method of inequalities”. In: *IEEE Transactions on Control Systems Technology* 2.4 (1994), pp. 455–461.

[79] T. Maeba et al. “Swing-up controller design for inverted pendulum by using energy control method based on Lyapunov function”. In: *The 2010 International Conference on Modelling, Identification and Control (ICMIC)*. 2010, pp. 768–773.

[80] L. Y. Wang and J. Zhang. “Fundamental limitations and differences of robust and adaptive control”. In: *American Control Conference, 2001. Proceedings of the 2001*. Vol. 6. 2001, 4802–4807 vol.6.

[81] N. H. El-Farra. “Integrating model-based fault detection and fault-tolerant control of distributed processes”. In: *American Control Conference, 2006*. 2006, pp. 6014–6019.

[82] R. Isermann, R. Schwarz, and S. Stolzl. “Fault-tolerant drive-by-wire systems”. In: *IEEE Control Systems* 22.5 (2002), pp. 64–81.

[83] W. H. Chung and J. L. Speyer. “A game theoretic fault detection filter”. In: *IEEE Transactions on Automatic Control* 43.2 (1998), pp. 143–161.

[84] L. H. Mutuel and J. L. Speyer. “A discrete-time game-theoretic fault detection filter”. In: *American Control Conference, 2000. Proceedings of the 2000*. Vol. 5. 2000, 3388–3392 vol.5.

[85] M. Nyberg and E. Frisk. “Residual Generation for Fault Diagnosis of Systems Described by Linear Differential-Algebraic Equations”. In: *IEEE Transactions on Automatic Control* 51.12 (2006), pp. 1995–2000.

[86] P. Mohajerin Esfahani et al. “A tractable nonlinear fault detection and isolation technique with application to the cyber-physical security of power systems”. In: *Decision and Control (CDC), 2012 IEEE 51st Annual Conference on*. 2012, pp. 3433–3438.

[87] S. Zhao and B. Huang. “Iterative Residual Generator for Fault Detection with Linear Time-Invariant State-Space Models”. In: *IEEE Transactions on Automatic Control* 62.10 (2017), pp. 5422–5428.

[88] G. Wang and Z. Huang. “Data-driven fault-tolerant control design for wind turbines with robust residual generator”. In: *IET Control Theory Applications* 9.7 (2015), pp. 1173–1179.

[89] J. Gertler and J. Cao. “PCA-based fault diagnosis in the presence of control and dynamics”. In: *AIChE Journal* 50.2 (2004), pp. 388–402.

[90] J. D. Isom and R. E. LaBarre. “Process fault detection, isolation, and reconstruction by principal component pursuit”. In: *American Control Conference (ACC), 2011*. 2011, pp. 238–243.

[91] J. Abraham, S. Banerjee, and A. Chatterjee. “Design of efficient error resilience in signal processing and control systems: From algorithms to circuits”. In: *2017 IEEE 23rd International Symposium on On-Line Testing and Robust System Design (IOLTS)*. 2017, pp. 192–195.

[92] S. Banerjee, A. Chatterjee, and J. A Abraham. “Checksum based error detection in linearized representations of non linear control systems”. In: *2016 17th Latin-American Test Symposium (LATS)*. IEEE. 2016, pp. 182–182.

[93] B. Friedland. *Control System Design An Introduction to State-space methods*. ISBN 0-486-44278-0. Dover Publications, 2005.

[94] R.W. Bass and I. Gura. "High-order system design via state-space considerations". In: *Proceedings of the Joint Automatic Control Conference*. June 1965, pp. 311–318.

[95] P. Cominos and N. Munro. "PID controllers: recent tuning methods and design to specification". In: *IEEE Proceedings of Control Theory and Applications*. Vol. 149. Issue 1. IEEE. Jan 2002, pp. 46–53.

[96] S. Banerjee, A. Chatterjee, and J. A. Abraham. "Real-time correction of dc servo motor and controller failures using analog checksums". In: *19th Annual International Mixed-Signals, Sensors, and Systems Test Workshop Proceedings*. 2014, pp. 1–5.

[97] M. I. Momtaz, S. Banerjee, and A. Chatterjee. "Real-time DC motor error detection and control compensation using linear checksums". In: *2016 IEEE 34th VLSI Test Symposium (VTS)*. 2016, pp. 1–6.

[98] G. D. Hachtel and F. Somensi. *Logic Synthesis and Verification Algorithms*. Springer, 2012.

[99] D. Perry and H. Foster. *Applied Formal Verification*. McGraw Hill, 2005.

[100] A. Papoulis. *Circuits and Systems: A Modern Approach*. 19th ed. United States: Oxford University Press, 1995.

[101] S. Banerjee et al. "Error Resilient Real-Time State Variable Systems for Signal Processing and Control". In: *Test Symposium (ATS), 2014 IEEE 23rd Asian*. 2014, pp. 39–44.

[102] S. Banerjee, A. Gomez-Pau, and A. Chatterjee. "Design of low cost fault tolerant analog circuits using real-time learned error compensation". In: *2014 19th IEEE European Test Symposium (ETS)*. 2014, pp. 1–2.

[103] A. Gomez-Pau, S. Banerjee, and A. Chatterjee. "Real-time transient error and induced noise cancellation in linear analog filters using learning-assisted adaptive analog checksums". In: *2014 IEEE 20th International On-Line Testing Symposium (IOLTS)*. 2014, pp. 25–30.

[104] T. Zhang, W.R. Eisenstadt, and Fox. R.M. "20 GHz bipolar RF RMS power detectors". In: *Bipolar/BiCMOS Circuits and Technology Meeting, 2005. Proceedings of the*. 9-11 Oct, 2005, pp. 204–207.

[105] H. G. Dimopoulos. *Analog Electronic Filters. Theory, Design and Synthesis*. New York: Springer, 2012.

[106] P. Shivakumar et al. "Modeling the effect of technology trends on the soft error rate of combinational logic". In: *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*. 2002, pp. 389–398.

[107] R.K. Iyer et al. "Recent advances and new avenues in hardware-level reliability support". In: *Micro, IEEE* 25.6 (2005), pp. 18–29.

[108] T. Karnik et al. “Scaling trends of cosmic ray induced soft errors in static latches beyond 0.18 μ”. In: *VLSI Circuits, 2001. Digest of Technical Papers. 2001 Symposium on*. 2001, pp. 61–62.

[109] Y.S. Dhillon, A.U. Diril, and A. Chatterjee. “Soft-error tolerance analysis and optimization of nanometer circuits”. In: *Design, Automation and Test in Europe, 2005. Proceedings*. 2005, 288–293 Vol. 1.

[110] B.W. Johnson. *Design and Analysis of Fault Tolerant Digital System*. Addison-Wesley, 1989.

[111] Y.Y. Tang, T. Li, and C.Y. Suen. “VLSI arrays for speech processing with linear predictive coding”. In: *Pattern Recognition, 1994. Vol. 3 - Conference C: Signal Processing, Proceedings of the 12th IAPR International Conference on*. 1994, 357–359 vol.3.

[112] A. Harma and U.K. Laine. “A comparison of warped and conventional linear predictive coding”. In: *Speech and Audio Processing, IEEE Transactions on* 9.5 (2001), pp. 579–588.

[113] R.C. Hendriks, R. Heusdens, and J. Jensen. “Perceptual linear predictive noise modelling for sinusoid-plus-noise audio coding”. In: *Acoustics, Speech, and Signal Processing, 2004. Proceedings. (ICASSP ’04). IEEE International Conference on*. Vol. 4. 2004, iv–189–iv–192 vol.4.

[114] W. Kinsner and E. Vera. “Fractal modelling of residues in linear predictive coding of speech”. In: *Cognitive Informatics, 2009. ICCI ’09. 8th IEEE International Conference on*. 2009, pp. 181–187.

[115] N. Wiener. “The Wiener RMS (Root Mean Square) Error Criterion in Filter Design and Prediction”. In: *Extrapolation, Interpolation, and Smoothing of Stationary Time Series:With Engineering Applications*. MIT Press, 1949, pp. 129–148. ISBN: 9780262257190.

[116] A.F. TIMAN. “Chapter II - The Best Approximation”. In: *Theory of Approximation of Functions of a Real Variable*. Ed. by A.F. TIMAN. International Series of Monographs on Pure and Applied Mathematics. Pergamon, 1963, pp. 26 –92.

[117] S. Boyd and L. Chua. “Fading memory and the problem of approximating nonlinear operators with Volterra series”. In: *IEEE Transactions on Circuits and Systems* 32.11 (1985), pp. 1150–1161.

[118] G. O. A. Glentis, P. Koukoulas, and N. Kalouptsidis. “Efficient algorithms for Volterra system identification”. In: *IEEE Transactions on Signal Processing* 47.11 (1999), pp. 3042–3057.

[119] G. A. Carpenter, S. Grossberg, and J. H. Reynolds. “ARTMAP: supervised real-time learning and classification of nonstationary data by a self-organizing neural network”. In: *[1991 Proceedings] IEEE Conference on Neural Networks for Ocean Engineering*. 1991, pp. 341–342.

[120] N. Binkert et al. “The Gem5 Simulator”. In: *SIGARCH Comput. Archit. News* 39.2 (Aug. 2011), pp. 1–7.

[121] *Pin - A Dynamic Binary Instrumentation Tool*. http://opencores.org/project,amber.

[122] M. Kaliorakis et al. "Differential Fault Injection on Microarchitectural Simulators". In: *Workload Characterization (IISWC), 2015 IEEE International Symposium on*. 2015, pp. 172–182.

[123] *Amber ARM-compatible core Overview*. https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool.

[124] K. Yoshida. "Swing-up control of an inverted pendulum by energy-based methods". In: *American Control Conference, 1999. Proceedings of the 1999*. Vol. 6. 1999, 4045–4047 vol.6.

[125] S. Banerjee et al. "Real-time checking of linear control systems using analog checksums". In: *2013 IEEE 19th International On-Line Testing Symposium (IOLTS)*. 2013, pp. 122–127.

[126] S. Anwar and B. Zheng. "An Antilock-Braking Algorithm for an Eddy-Current-Based Brake-By-Wire System". In: *IEEE Transactions on Vehicular Technology* 56.3 (2007), pp. 1100–1107.

[127] U. Kiencke and L. Nielsen. *Automotive Control Systems*. Springer-Verlag Berlin Heidelberg, 2005.

[128] Pololu Robotics and Electronics. "Balboa 32U4 Balancing Robot Kit". In: *https://goo.gl/FUJeEW* ().

[129] P. J. Werbos. "Neural Networks for Control". In: Cambridge, MA, USA: MIT Press, 1990. Chap. A Menu of Designs for Reinforcement Learning over Time, pp. 67–95. ISBN: 0-262-13261-3.

[130] Q. Wei and D. Liu. "Data-Driven Neuro-Optimal Temperature Control of Water-Gas Shift Reaction Using Stable Iterative Adaptive Dynamic Programming". In: *IEEE Transactions on Industrial Electronics* 61.11 (2014), pp. 6399–6408.

[131] H. Modares and F. L. Lewis. "Optimal Tracking Control of Nonlinear Partially-unknown Constrained-input Systems Using Integral Reinforcement Learning". In: *Automatica* 50.7 (July 2014), pp. 1780–1792.

[132] I. Grondman et al. "A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42.6 (2012), pp. 1291–1307.

[133] R. Song et al. "Off-Policy Actor-Critic Structure for Optimal Control of Unknown Systems With Disturbances". In: *IEEE Transactions on Cybernetics* 46.5 (2016), pp. 1041–1050.

[134] B. Wang et al. "Design and implementation of an adaptive cruise control system based on supervised actor-critic learning". In: *2015 5th International Conference on Information Science and Technology (ICIST)*. 2015, pp. 243–248.

[135] I. S. Comşa et al. “Adaptive proportional fair parameterization based LTE scheduling using continuous actor-critic reinforcement learning”. In: *2014 IEEE Global Communications Conference*. 2014, pp. 4387–4393.

[136] R. Li et al. “TACT: A Transfer Actor-Critic Learning Framework for Energy Saving in Cellular Radio Access Networks”. In: *IEEE Transactions on Wireless Communications* 13.4 (2014), pp. 2000–2011.

[137] I. Grondman et al. “Efficient Model Learning Methods for Actor Critic Control”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 42.3 (2012), pp. 591–602.

[138] A. Y. Ng et al. “Autonomous Inverted Helicopter Flight via Reinforcement Learning”. In: *Experimental Robotics IX: The 9th International Symposium on Experimental Robotics*. Ed. by Marcelo H. Ang and Oussama Khatib. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 363–372. ISBN: 978-3-540-33014-1.

[139] K. Doya. “Reinforcement Learning in Continuous Time and Space”. In: *Neural Comput.* 12.1 (Jan. 2000), pp. 219–245.

[140] V. Konda. “Actor-critic Algorithms”. AAI0804543. PhD thesis. Cambridge, MA, USA, 2002.

[141] C. J. C. H. Watkins. “Learning from Delayed Rewards”. PhD thesis. Cambridge, UK: King’s College, 1989.

[142] C. Watkins and P. Dayan. “Q-learning”. In: *Machine Learning* 8 (1992), pp. 279–292.

[143] S. J. Bradtke, B. E. Ydstie, and A. G. Barto. “Adaptive linear quadratic control using policy iteration”. In: *American Control Conference, 1994*. Vol. 3. 1994, 3475–3479 vol.3.

[144] G. A. Rummery and M. Niranjan. *On-Line Q-Learning Using Connectionist Systems*. Tech. rep. 1994.

[145] R. S. Sutton and A. G. Barto. “Generalization and Function Approximation”. In: *Reinforcement Learning:An Introduction*. MIT Press, 1998, pp. 193–226. ISBN: 9780262257053.

[146] V. Gullapalli. *A Stochastic Algorithm for Learning Real-valued Functions via Reinforcement*. Tech. rep. Amherst, MA, USA, 1988.

[147] R. J. Williams. “Simple statistical gradient-following algorithms for connectionist reinforcement learning”. In: *Machine Learning* 8.3 (1992), pp. 229–256.

[148] J. Peters and S. Schaal. “2008 Special Issue: Reinforcement Learning of Motor Skills with Policy Gradients”. In: *Neural Netw.* 21.4 (May 2008), pp. 682–697.

[149] V. R. Konda and J. N. Tsitsiklis. “OnActor-Critic Algorithms”. In: *SIAM Journal on Control and Optimization* 42.4 (2003), pp. 1143–1166. eprint: `http://dx.doi.org/10.1137/S0363012901385691`.

[150] J. Peters, K. Mülling, and Y. Altün. “Relative Entropy Policy Search”. In: *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*. AAAI’10. Atlanta, Georgia: AAAI Press, 2010, pp. 1607–1612.

[151] A. G. Barto, R. S. Sutton, and C. W. Anderson. “Neuronlike adaptive elements that can solve difficult learning control problems”. In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-13.5 (1983), pp. 834–846.

[152] J. Peters and S. Schaal. “Natural Actor-Critic”. In: *Neurocomputing* 71.79 (2008), pp. 1180 –1190.

[153] S. Bhatnagar et al. “Natural actorcritic algorithms ”. In: *Automatica* 45.11 (2009), pp. 2471 –2482.

[154] S. Adam, L. Busoniu, and R. Babuska. “Experience Replay for Real-Time Reinforcement Learning Control”. In: *Trans. Sys. Man Cyber Part C* 42.2 (Mar. 2012), pp. 201–212.

[155] J. C. Kinsey, Q. Yang, and J. C. Howland. “Nonlinear Dynamic Model-Based State Estimators for Underwater Navigation of Remotely Operated Vehicles”. In: *IEEE Transactions on Control Systems Technology* 22.5 (2014), pp. 1845–1854.

[156] M. I. Momtaz, S. Banerjee, and A. Chatterjee. “On-line diagnosis and compensation for parametric failures in linear state variable circuits and systems using time-domain checksum observers”. In: *2017 IEEE 35th VLSI Test Symposium (VTS)*. 2017, pp. 1–6.

[157] M. R. Buehner et al. “Improving performance using robust recurrent reinforcement learning control”. In: *2007 European Control Conference (ECC)*. 2007, pp. 1676–1681.

[158] S. Banerjee and A. Chatterjee. “Real-time self-learning for control law adaptation in nonlinear systems using encoded check states”. In: *2017 22nd IEEE European Test Symposium (ETS)*. 2017, pp. 1–6.

[159] A.Abur and A.G.Exposito. *Power System State Estimation: Theory and Implementation*. CRC Press, 2004.

[160] A. Teixeira et al. “Cyber security analysis of state estimators in electric power systems”. In: *49th IEEE Conference on Decision and Control (CDC)*. 2010, pp. 5991–5998.

[161] Y. Chakhchoukh and H. Ishii. “Enhancing Robustness to Cyber-Attacks in Power Systems Through Multiple Least Trimmed Squares State Estimations”. In: *IEEE Transactions on Power Systems* 31.6 (2016), pp. 4395–4405.

[162] G. Valverde et al. “A Constrained Formulation for Hybrid State Estimation”. In: *IEEE Transactions on Power Systems* 26.3 (2011), pp. 1102–1109.

[163] M. Asprou, E. Kyriakides, and M. Albu. “The Effect of Variable Weights in a WLS State Estimator Considering Instrument Transformer Uncertainties”. In: *IEEE Transactions on Instrumentation and Measurement* 63.6 (2014), pp. 1484–1495.

[164] R. Sodhi, S. C. Srivastava, and S. N. Singh. “Phasor-assisted Hybrid State Estimator”. In: *Electric Power Components and Systems* 38.5 (2010), pp. 533–544. eprint: http://dx.doi.org/10.1080/15325000903376925.

[165] M. Lixia et al. “A software-only PTP synchronization for power system state estimation with PMUs”. In: *2011 IEEE International Instrumentation and Measurement Technology Conference*. 2011, pp. 1–6.

[166] M. Pau, P. A. Pegoraro, and S. Sulis. “Efficient Branch-Current-Based Distribution System State Estimation Including Synchronized Measurements”. In: *IEEE Transactions on Instrumentation and Measurement* 62.9 (2013), pp. 2419–2429.

[167] J. Liu et al. “Trade-Offs in PMU Deployment for State Estimation in Active Distribution Grids”. In: *IEEE Transactions on Smart Grid* 3.2 (2012), pp. 915–924.

[168] J. A. Lopez and C. N. Lu. “Phasor Measurement Unit placement for wide area instability detection and prediction”. In: *2017 IEEE 3rd International Future Energy Electronics Conference and ECCE Asia (IFEEC 2017 - ECCE Asia)*. 2017, pp. 1708–1713.

[169] M. H. F. Wen and V. O. K. Li. “Optimal Phasor Data Compression Unit Installation for Wide-Area Measurement Systems - An Integer Linear Programming Approach”. In: *IEEE Transactions on Smart Grid* 7.6 (2016), pp. 2644–2653.

[170] S. Banerjee, A. Chatterjee, and J. A. Abraham. “Efficient cross-layer concurrent error detection in nonlinear control systems using mapped predictive check states”. In: *2016 IEEE International Test Conference (ITC)*. 2016, pp. 1–10.

[171] S. Amin, A. A. Cárdenas, and S. S. Sastry. “Safe and Secure Networked Control Systems Under Denial-of-Service Attacks”. In: *Proceedings of the 12th International Conference on Hybrid Systems: Computation and Control*. HSCC ’09. San Francisco, CA: Springer-Verlag, 2009, pp. 31–45. ISBN: 978-3-642-00601-2.

[172] Y. Liu, P. Ning, and M. K. Reiter. “False Data Injection Attacks Against State Estimation in Electric Power Grids”. In: *Proceedings of the 16th ACM Conference on Computer and Communications Security*. CCS ’09. Chicago, Illinois, USA: ACM, 2009, pp. 21–32.

[173] G. Hug and J. A. Giampapa. “Vulnerability Assessment of AC State Estimation With Respect to False Data Injection Cyber-Attacks”. In: *IEEE Transactions on Smart Grid* 3.3 (2012), pp. 1362–1370.

[174] M. Esmalifalak et al. “Stealth false data injection using independent component analysis in smart grid”. In: *2011 IEEE International Conference on Smart Grid Communications (SmartGridComm)*. 2011, pp. 244–248.

[175] M. A. Rahman and H. Mohsenian-Rad. “False data injection attacks with incomplete information against smart power grids”. In: *2012 IEEE Global Communications Conference (GLOBECOM)*. 2012, pp. 3153–3158.

[176] J. Liang, O. Kosut, and L. Sankar. “Cyber attacks on AC state estimation: Unobservability and physical consequences”. In: *2014 IEEE PES General Meeting — Conference Exposition*. 2014, pp. 1–5.

[177] Y. Mo and B. Sinopoli. “Secure control against replay attacks”. In: *2009 47th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. 2009, pp. 911–918.

[178] Y. Mo et al. “Cyber-Physical Security of a Smart Grid Infrastructure”. In: *Proceedings of the IEEE* 100.1 (2012), pp. 195–209.

[179] R. S. Smith. “A Decoupled Feedback Structure for Covertly Appropriating Networked Control Systems”. In: *IFAC Proceedings Volumes* 44.1 (2011). 18th IFAC World Congress, pp. 90 –95.

[180] H. Sandberg, A. Teixeira, and K. H. Johansson. “On Security Indices for State Estimators in Power Networks”. In: *Preprints of the First Workshop on Secure Control Systems, CPSWEEK 2010, Stockholm, Sweden*. QC 20120206. 2010.

[181] O. Kosut et al. “Malicious Data Attacks on the Smart Grid”. In: *IEEE Transactions on Smart Grid* 2.4 (2011), pp. 645–658.

[182] M. A. Rahman and H. Mohsenian-Rad. “False data injection attacks against nonlinear state estimation in smart power grids”. In: *2013 IEEE Power Energy Society General Meeting*. 2013, pp. 1–5.

[183] J. Kim and L. Tong. “On Topology Attack of a Smart Grid: Undetectable Attacks and Countermeasures”. In: *IEEE Journal on Selected Areas in Communications* 31.7 (2013), pp. 1294–1305.

[184] J. Zhang and L. Sankar. “Physical System Consequences of Unobservable State-and-Topology Cyber-Physical Attacks”. In: *IEEE Transactions on Smart Grid* 7.4 (2016), pp. 2016–2025.

[185] L. Schenato et al. “Foundations of Control and Estimation Over Lossy Networks”. In: *Proceedings of the IEEE* 95.1 (2007), pp. 163–187.

[186] F. Pasqualetti, F. Dorfler, and F. Bullo. “Attack Detection and Identification in Cyber-Physical Systems”. In: *IEEE Transactions on Automatic Control* 58.11 (2013), pp. 2715–2729.

[187] F. Pasqualetti, F. Dorfler, and F. Bullo. “Control-Theoretic Methods for Cyber-Physical Security: Geometric Principles for Optimal Cross-Layer Resilient Control Systems”. In: *IEEE Control Systems Magazine* 35.1 (2015), pp. 110–127.

[188] H. Fawzi, P. Tabuada, and S. Diggavi. “Secure Estimation and Control for Cyber-Physical Systems Under Adversarial Attacks”. In: *IEEE Transactions on Automatic Control* 59.6 (2014), pp. 1454–1467.

[189] R. B. Bobba et al. “Detection False data injection attacks on DC state estimation”. In: *Proceedings of 1st Workshop on Secure Control Systems (SCS)*. 2010, pp. 1–9.

[190] L. Liu et al. “Detecting False Data Injection Attacks on Power Grid by Sparse Optimization”. In: *IEEE Transactions on Smart Grid* 5.2 (2014), pp. 612–621.

[191] S. Li, Y. Ylmaz, and X. Wang. “Quickest Detection of False Data Injection Attack in Wide-Area Smart Grids”. In: *IEEE Transactions on Smart Grid* 6.6 (2015), pp. 2725–2735.

[192] S. Gao et al. “Automated vulnerability analysis of AC state estimation under constrained false data injection in electric power systems”. In: *2015 54th IEEE Conference on Decision and Control (CDC)*. 2015, pp. 2613–2620.

[193] M. Cosovic and D. Vukobratovic. “Distributed Gauss-Newton method for AC state estimation: A belief propagation approach”. In: *2016 IEEE International Conference on Smart Grid Communications (SmartGridComm)*. 2016, pp. 643–649.

[194] A. Monticelli. *State Estimation in Electric Power Systems, A Generalized Approach*. Kluwer Academic Publishers, 1999.

[195] A. Wood and B. Wollenberg. *Power generation, operation, and control*. John Wiley and Sons, 1996.

[196] A. ABur and A. G. Exposito. *Power System State Estimation: Theory and Implementation*. Boca Raton, FL: CRC Press, 2004.

[197] R. D. Zimmerman, C. E. Murillo-Sanchez, and R. J. Thomas. “MATPOWER’s extensible optimal power flow architecture”. In: *2009 IEEE Power Energy Society General Meeting*. 2009, pp. 1–7.

[198] R. DeCarlo. *Linear Systems:A State Variable Approach with Numerical Implementation*. Prentice Hall, 1989.

[199] V. Guruswami, J. R. Lee, and A. Wigderson. “Euclidean Sections of ℓ_1^N with Sublinear Randomness and Error-Correction over the Reals”. In: *Proceedings of RANDOM ’08* (2008), pp. 444–454.

[200] E. J. Candes and T. Tao. “Decoding by linear programming”. In: *IEEE Transactions on Information Theory* 51.12 (2005), pp. 4203–4215.

[201] S. Sundaram and C. N. Hadjicostis. “Distributed Function Calculation via Linear Iterative Strategies in the Presence of Malicious Agents”. In: *IEEE Transactions on Automatic Control* 56.7 (2011), pp. 1495–1508.

[202] B. Molinari. “Extended controllability and observability for linear systems”. In: *IEEE Transactions on Automatic Control* 21.1 (1976), pp. 136–137.

[203] F. Sandhu et al. “FPGA-Based Implementation of Kalman Filter for Real-Time Estimation of Tire Velocity and Acceleration”. In: *IEEE Sensors Journal* 17.17 (2017), pp. 5749–5758.

[204] M. Ricco et al. “FPGA-Based Implementation of Dual Kalman Filter for PV MPPT Applications”. In: *IEEE Transactions on Industrial Informatics* 13.1 (2017), pp. 176–185.

[205] D. Pritsker. “Hybrid implementation of Extended Kalman Filter on an FPGA”. In: *2015 IEEE Radar Conference (RadarCon)*. 2015, pp. 0077–0082.

[206] X. Jiang, J. Tao, and Z. Zhang. “Implementation of extended Kalman Filter on FPGA”. In: *IET International Conference on Communication Technology and Application (ICCTA 2011)*. 2011, pp. 915–921.

[207] C. Josz et al. “AC Power Flow Data in MATPOWER and QCQP Format: iTesla, RTE Snapshots, and PEGASE”. In: *ArXiv e-prints* (Mar. 2016). arXiv: 1603.01533 [math.OC].

[208] S. Fliscounakis et al. “Contingency Ranking With Respect to Overloads in Very Large Power Systems Taking Into Account Uncertainty, Preventive, and Corrective Actions”. In: *IEEE Transactions on Power Systems* 28.4 (2013), pp. 4909–4917.

[209] M. Grant and S. Boyd. *CVX: Matlab Software for Disciplined Convex Programming, version 2.1*. http://cvxr.com/cvx. Mar. 2014.

[210] S. Pandey, S. Banerjee, and A. Chatterjee. “Concurrent error detection and tolerance in Kalman filters using encoded state and statistical covariance checks”. In: *2016 IEEE 22nd International Symposium on On-Line Testing and Robust System Design (IOLTS)*. 2016, pp. 161–166.